CODEN:LUTEDX/(TEIE-3072)/1-42/(2017)

Electrical Engineering and Automation

Industrial

Automation av analysorderhantering

Dennis Hinz
Axel Mattsson

Division of Industrial Electrical Engineering and Automation
Faculty of Engineering, Lund University

2017

Automation av analysorderhantering

LUNDS UNIVERSITET
Campus Helsingborg

LTH Ingenjörshögskolan vid Campus Helsingborg
Avdelningen för Industriell Elektroteknik och Automation

DENNIS HINZ & AXEL MATTSSON

LTH Ingenjörshögskolan vid Campus Helsingborg
Lunds universitet
Box 882
251 08 Helsingborg

LTH School of Engineering
Lund University
Box 882
SE-251 08 Helsingborg
Sweden

Tryckt i Sverige
Lunds universitet
Lund 2017

Sammanfattning

Byggnader som byggs och asfalt som läggs måste testas så att de uppfyller vissa krav och standarder. Dessa tester görs på Peabs teknikcenter/laboratorium och det är just på ett av deras laboratorium i Helsingborg som detta examensarbete är utfört.

När laboratoriet får ett betongprov ska den tillhörande analysordern registreras i Peabs labbprogram för att sedan kunna få ut rätt blanketter för den specificerade provningen. Registreringen har tidigare skett manuellt genom att direkt skriva av analysordern som kunden skickat in. Detta har alltså inneburit dubbelarbete och det är detta som elimineras av webbapplikationen som har utvecklats under examensarbetet.

Lösningen består av en webbapplikation där beställaren kan lägga sina analysordrar, en databas där information kan lagras och ett prototypprogram för att visa ett exempel på hur det kan se ut. Webbapplikationen är skapad tillsammans med databasen i programutvecklingsmiljön Visual Studio som är en utvecklingsmiljö skapad av Microsoft. Prototypprogrammet är skapat i Eclipse med Java för att illustrera hur den nya webbapplikationen för orderläggning kan samverka med det program som laboratoratorieteknikerna använder genom att all information om en order direkt hämtas från databasen då en sökning efter dess orderid görs i prototypprogrammet.

Tack vare denna förenklade process att registrera prover försvinner dubbelarbete från både de som lägger ordrarna och personalen på laboratoriet som skriver in informationen. Mänskliga fel såsom inmatning av fel information från laboratorieteknikerna försvinner då de hämtar en direkt kopia av beställarens order. Tiden som sparas via applikationen är beräknad till totalt 7 minuter per analysorder och under ett år inkommer ungefär 2000 ordrar.

Nyckelord

Peab, databas, programmering, analysorder, webbapplikation.

Abstract

Buildings that are built and the asphalt that is used for the paving needs to be tested to see if they fulfill certain requirements and standards. These types of tests are performed at Peab's technology centers/laboratories and it is in one of their labs in Helsingborg that the degree project has been executed.

As soon as the laboratory gets a concrete sample, they are supposed to register it in their computer software to be able to get the correct forms for the test. The registration process has previously been handled manually by copying the information about the concrete straight of the analysis order that's been placed by the customer. This has meant lots of duplication work and this is what is eliminated by using the web application developed in the degree project.

The solution consists of a web application where the customers can submit their analysis orders, a database where the information can be stored and a prototype program to show how the cooperation could work. The web application is developed together with the database in a program called Visual Studio which is created by Microsoft. A prototype program is also developed in Eclipse with Java coding to illustrate how the new web application for order submittal can interact with the application used by the laboratory technicians by immediately extracting all information about an order from the database when its orderid is searched for in the prototype program.

Thanks to the simplified process that is created for the registration of samples, the work duplication from the ones who put the orders and the personnel at the laboratory who copies the information straight from the customer's analysis order. Human mistakes such as the input of incorrect information in the program by the laboratory technicians are completely eliminated because they are now a complete duplicate of the analysis order. Time saved by using this application is about seven minutes in total per analysis order.

Keywords

Peab, database, programming, analysis order, web application.

Innehållsförteckning

Förord

Denna rapport är en del av examensarbetet för Axel Mattsson och Dennis Hinz på *Elektroteknik med automation* på LTH Campus Helsingborg. Det har varit ett utmanande och lärorikt arbete där vi samarbetat med flera olika företag. Storleken på arbetet är 22,5 högskolepoäng som motsvarar 600 timmar per person under läsperiod 6.

Vi skulle vilja tacka Peab Asfalt och de anställda på laboratoriet i Helsingborg för ett gott samarbete och för att de lät oss både göra vårt projekt i automation och examensarbetet hos dem. Ett speciellt tack till handledaren Olof Åkesson på Peab som lagt ner väldigt mycket tid på att hjälpa oss med kontakt med andra inblandade företag och tålmodigt lyssnat på våra ideér. Vi vill även tacka handledare Christian Nyberg och examinator Mats Lilja som hjälpt och väglett oss till att komma fram till hur arbetet ska byggas upp och för den kunskap vi fått från de kurser i vilka ni har varit våra föreläsare.

1. Inledning

1.1 Bakgrund

Arbetet utförs i samarbete med Peab Asfalt. Peab Asfalt är ett dotterbolag inom Peab-koncernen som är ett av Sveriges 50 största företag räknat i omsättning såväl som i antal anställda. Peab AB är verksamma inom områdena bygg, anläggning, industri och projektutveckling och har sitt huvudkontor i Förslöv i nordvästra Skåne. Grundarna av Peab var två bröder, Erik och Mats Paulsson.

Peab Asfalt har ett laboratorium i Helsingborg som utför prover på asfalt, betong, grus och jord. Arbetet kommer att förenkla processen för registrering av nyinkomna analysordrar av betongprover. De vanligaste typer av prover som utförs på betongen är tryckprover och frysprover. Ett tryckprov utförs genom att utsätta provkroppen för en tryckkraft. När kraften får provkroppen att gå sönder får man resultatet av provningen. Frysprov utspelas oftast under 56 dygn. Under dessa 56 dygn står provkroppen i en frys vars temperatur som varierar i intervallet -20 till +20 grader för att simulera miljön som betongen kommer utsättas för. Dessa prover görs för att säkerställa att betongen håller för det den är tänkt för. Innan proverna kan starta måste laboratorieteknikerna registrera information om provobjektet som ska analyseras. Under ett år får laboratoriet in cirka 2000 analysordrar. I dagsläget skrivs inkommen analysorder in manuellt i ett labbprogram. Varje fält måste då fyllas i två gånger, både av beställaren och av laboratorieteknikern på Peab. Detta innebär dubbelt arbete från två håll och risk för felinmatning på laboratoriet. En ny metod för inmatning ska skapas för att förbättra ovanstående. Hur den nuvarande metoden ser ut visas i fig 1.1.

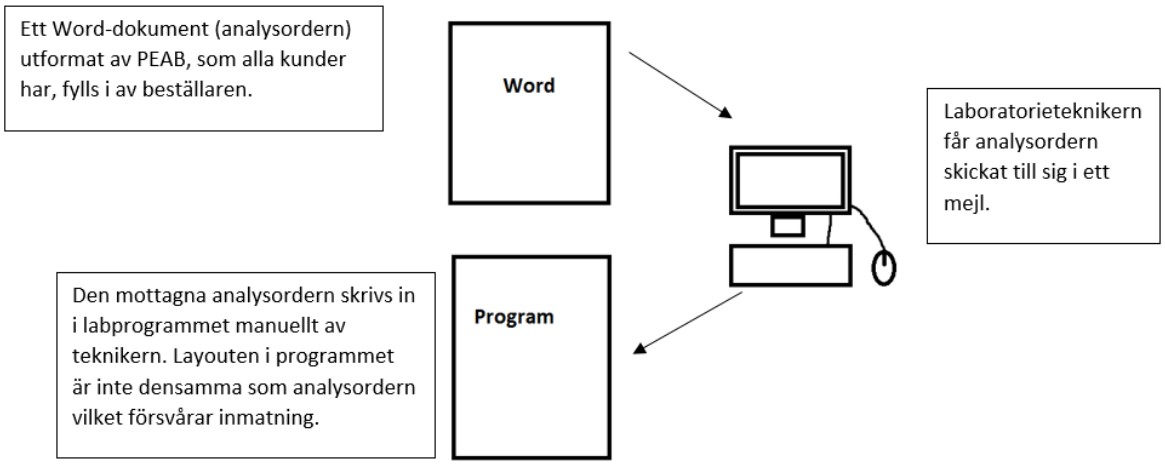


Fig 1.1. Nuvarande metod av inmatning från laboratoriets sida.

1.2 Syfte

Avsikten med arbetet är att förenkla och effektivisera registreringsprocessen av analysordrar. Automatiseringen ska även eliminera risken för felinmatningar vid manuell inmatning. Då orderläggningen sker över internet måste säkerheten säkerställas så att obehöriga inte får åtkomst till företagets handlingar.

För att undvika fel vid orderläggningen ska vissa fält göras obligatoriska och en del ska även kontrolleras så att de är rimliga.

En prototyp ska utvecklas som kan användas som underlag för vidareutveckling av de olika labbprogram som används idag av teknikerna på Peab Asfalt AB och av Swerock.

Om tid finns ska det på varje orderutskrift finnas en streckkod/QR-kod innehållande nyckeln till ordern så att en optisk läsare kan användas för snabb inmatning i labbprogrammet då ordern har skrivits ut. En funktion för detta ska i så fall också läggas till i vårt labbprogram.

En skiss över hur det nya arbetsflödet är tänkt att fungera kan ses i fig 1.2.

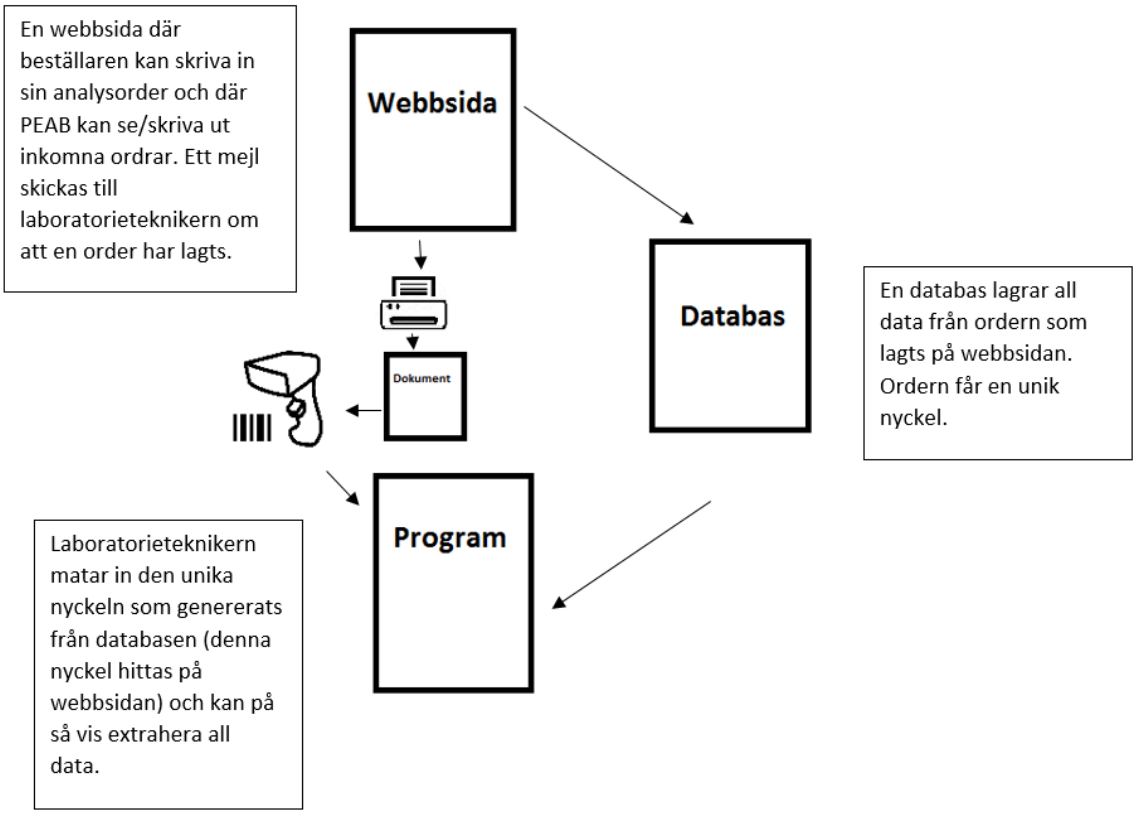


Fig 1.2. Planerat arbetsflöde efter implementering av webbapplikation.

1.3 Målformulering

En lösning ska utvecklas som inkluderar: en webbsida där kunden loggar in och lägger en order med uppgifter om beställaren samt information om betongen som ska analyseras; en databas som lagrar all information om ordern; ett prototypprogram som innehåller en del av de funktioner som finns hos originalprogrammet som idag används av laboratorieteknikerna.

Varje order ska få en nyckel som är kopplat till alla övriga attribut i databasen, denna ska sedan matas in i labbprogrammet, varpå all information hämtas från databasen.

Med admin-inloggning ska tillgång till samtliga tidigare ordrar ges och dessa ska därmed kunna skrivas ut. Med kundinloggning kan den specifika kunden skriva ut sina tidigare lagda ordrar.

Målet är att helt eliminera det dubbelarbete som just nu pågår. Man ska inte behöva skriva av analysordern. Labbprogrammet ska ha direkt tillgång till den information som beställaren matat in. Ett annat mål är att eliminera risken för felinmatning från laboratorieteknikerns sida.

För att uppnå ett användarvänligt webbinterface för orderläggning och ett enkelt program ska olika förslag utvecklas som sedan företagen får utvärdera.

1.4 Problemformulering

Planeringen av examensarbetet utgår från följande frågor och kräver att dessa besvaras:

- Kan hemsidan som ska skapas göras tillräckligt säker?
- Fyller webbsidan PEAB IT:s krav angående säkerhet och utvecklingsmiljö?
- Hur ska webbinterfacet utformas för att vara så användarvänligt som möjligt?
- Hur ska vi implementera en funktion för extrahering av data från databasen till det nya labbprogrammet som ska skapas?
- Vilka programmeringsspråk ska användas?
- Hur ska databasen utformas för att kunna samarbeta med både en webbsida och ett program?
- Hur ska vår prototyp integreras i de befintliga programmen?
- Kan vi lägga in en funktion i programmet som med hjälp av streckkodsläsare kan skanna in utskrivna ordrar?

1.5 Motivering av examensarbetet

En i gruppen har tidigare arbetat på arbetsplatsen och har därmed stor insikt i hur arbetet utförs och vilka effektiviseringar som kan göras. Den effektivisering som valts känner vi att har stort samband med det vi har lärt oss under utbildningen. Varför det valts att göra en prototyp istället för ett fullständigt, lanserbart program är på grund av att PEAB vill implementera denna typ av funktion i två av deras nuvarande laboratorieprogram, varav det ena programmet ägs av Peab medan det andra är ett program som de bara äger licens till. Prototypen ska sedan kunna implementeras i de nuvarande programmen som används.

1.6 Avgränsningar

Laboratorieprogrammet skall inte innehålla all funktionalitet hos det program som idag används utan är bara ett prototypprogram som används för att illustrera hur webbapplikation och laborationsprogram kan interagera. Programmet utvecklas för enbart användning i Windows OS. Hemsidan kommer inte att anpassas för mobiler. Åtkomsten av data ska begränsas för kunderna så att de bara kan se sina egna order.

2. Teknisk bakgrund

Detta kapitel beskriver den teknik som använts under examensarbetet och ger den information som krävs för förståelse av hur projektet har utvecklats.

2.1 XML

XML är en dokumenttyp som liksom HTML använder sig av taggar (se fig. 2.1). Till skillnad från HTML kan taggarna dock skapas av användaren själv för att ge mening till data. Med hjälp av dessa taggar kan man lätt extrahera ut information från dokumentet[1]. XML utvecklades av en grupp som bestod av elva personer. Den första versionen av XML blev färdigutvecklad 1998 [7].

För att användning av XML som extraheringsmetod skall fungera måste man veta namnen på alla de taggar som finns i dokumentet. Denna typ av extraheringsmetod användes på webbsidan för att en användare ska kunna hämta receptet direkt från tillverkarens program *SIMMA*. Ett exempel på hur ett sådant XML-dokument kan se ut ses i fig 2.1.

```
<?xml version="1.0" encoding="UTF-8"?>
<recept>
  <märkning>Märkning</märkning>
  <gjutdatum>2017-04-06</gjutdatum>
  <ålder>10</ålder>
  <leverantör>Leverantör</leverantör>
  <fabriksnr>1</fabriksnr>
</recept>
```

Fig 2.1. Exempel på hur ett XML-dokument kan se ut.

2.2 ASP.NET

ASP.NET är en vidareutveckling av det äldre ASP (Active Server Pages). Det är ett ramverk som tillhandahåller en struktur för enklare utveckling av dynamiska webbsidor med en mängd olika kontroller såsom textrutor, knappar, validerare för obligatoriska fält, för att nämna några. Dessutom tillhandahålls en stor mängd standardfunktioner för att manipulera dessa kontroller, hantera olika *events*, med mera. Utvecklingen är uppdelad i en design-del, i vilken HTML/CSS används för att specificera vilket innehåll sidan ska ha och hur det ska presenteras, och en *code-behind*-del i vilken något programspråk används för att bestämma logiken bakom hur applikationen ska bete sig. Vanligtvis används en fil för presentationen med filtillägget .aspx och en annan för logiken med filtillägget .cs om utveckling sker med C#.

Bland de programspråk som finns tillgängliga för ASP.NET är C#, VB.NET och J#. ASP.NET använder sig av *Common Language Runtime* (*CLR*) som i likhet med Javas JVM är en *virtual machine* som möjliggör exekvering på alla operativsystem.

Källkoden omvandlas först till *Common Intermediate Language (CIL)*, därefter kompileras den till byte-kod som antingen tolkas och exekveras av virtual machine eller kompileras till maskinkod med hjälp av en *Just In Time-kompilator*. Dessutom tillhandahålls liksom med JVM tjänster för bland annat minneshantering, skräpinsamling och exceptionhantering.

Detta ramverk är utvecklat av Microsoft och det är därför lättare att synkronisera med andra Microsoft-program. ASP.NET används i utvecklingsmiljön Visual Studio. Att använda detta ramverk var ett krav från Peab IT.

2.3 Transact-SQL

Transact-SQL, även kallat T-SQL, är kommunikationsmedlet mellan databasen och webbapplikationen. Detta programspråk är utvecklat av Microsoft och är en utvidgning av SQL. Klausulerna som används i T-SQL är desamma som används i MySQL såsom *Select*, *From* och *Where*. Vissa skillnader förekommer dock i syntaxen där bland annat hakparanteser används runt tabellnamn. En del funktioner skiljer sig dessutom från MySQL vad gäller namn och andra funktioner som finns tillgängliga i MySQL saknas här helt. T-SQL tillåter användning av en try-catch sats för de errors som kan kastas. Det tillåter dessutom konditionella uttryck i form av if-satser och loopar med while-satser.

2.4 Visual Studio

Visual Studio är en *Integrated Development Environment (IDE)*, det vill säga integrerad programutvecklingsmiljö, utvecklad av Microsoft. Med denna utvecklingsmiljö kan både datorprogram och webbsidor skapas. Den tillhandahåller en kodeditor, *debugger* och kompilator. Med *IntelliSense* ges dessutom ett väldigt bra stöd genom kodkomplettering vilket såväl påskyndar kodningen genom att bespara knapptryckningar som informerar om vilka funktioner som finns tillgängliga, vilket underlättar förståelsen.

I Visual Studio finns ett gott stöd för utveckling av webbapplikationers alla komponenter. För presentationen av webbsidan finns det ett läge som startas när en aspx-fil öppnas. I detta läge finns det en *Source*-flik där HTML-/CSS-kod används och en *Design*-flik i vilket resultatet kan inspekteras och där visst designarbete kan göras direkt med *drag and drop* av kontroller. Öppnas en cs-fil startas istället ett läge där C# (eller annat programspråk) används för att koda logiken.

Visual Studio har även ett brett stöd för design av SQL-databas.

Tack vare *Common Language Runtime* går det att välja vilket språk man vill programmera i. Koden kompileras inte direkt till maskinspråk till skillnad från andra språk utan kompileras till *Common Intermediate Language* som är ett objektorienterat assemblyspråk. Bland alternativen för programmeringsspråk finns Visual Basic, Visual C# och Visual C++.

Programvaran finns tillgänglig i tre versioner: Community, Professional och Enterprise. Detta projekt använde sig av Community version som är en gratisversion.

2.5 JavaScript

JavaScript är ett högnivåspråk som oftast används i webbtillämpningar. Det är ett dynamiskt och svagt typat skriptspråk som används för att ge webbsidor ett mått av interaktivitet [5][6]. Att det är svagt typat innebär att man istället för att ange datatyper för en variabel (int, string, double etc.) anges istället endast att det är en variabel. Detta görs genom att deklarera den med *var*.

Istället för att låta logiken ske på serversidan, som annars sker i ASP.NET, kan JavaScript användas för att utföra logiken på klientsidan. Det kan till exempel användas för att kontrollera att ett skrivfält på webbsidan är ifyllt utan att sidan måste skickas tillbaka för processering av servern. Javascript har ingenting med programmeringsspråket Java att göra.

De flesta webbläsarna kan tolka JavaScript vilket gör att de flesta som skapar webbapplikationer använder sig av detta högnivåspråk.

JavaScriptet inkluderas i HTML-koden under en <Script>-tag. Antingen skrivs hela JavaScriptet direkt under script-taggen eller så används en separat fil med filtillägget .js som sedan hänvisas till med hjälp av src-attributet.

2.6 C#

För programmering av webbapplikationen användes C#, som är ett objektorienterat högnivåspråk utvecklat av Microsoft [8] som en del av .NET. Språket liknar till stor del Java både vad gäller syntax och att det använder en virtuell maskin och skräpsamlare, men det är officiellt utvecklat med C++ som grund och har liksom C++ pekare.

2.7 Microsoft Word

Word-dokument består av en ZIP-fil innehållande XML-filer. I dessa XML-filer skapas taggar som används för att ge mening åt data. Dessa taggar kan sedan utnyttjas för att extrahera specifika data eller för att infoga data.

2.8 AJAX

AJAX står för *Asynchronous JavaScript and XML* och är en teknik för interaktion mellan webbapplikation och server. Detta möjliggör att data kan skickas fram och tillbaka mellan server och webbsida utan att sidan behöver laddas om (t.ex. uppdatering av data på webbsidan, skicka data till server i bakgrunden). Med Visual Studio kommer några AJAX Extensions förinstallerade och det finns även ett mer extensivt *AJAX Control Toolkit* tillgängligt för nedladdning från *CodePlex* [10].

3. Metod

Arbetet har under hela arbetsgången utgått från gantt-schemat som konstruerades när projektbeskrivningen skrevs. Men då flera företag har blivit inblandade efter hand har det krävts mer arbete än planerat.

3.1 Planering

Planering av arbetet gjordes genom att diskussion fördes med dåvarande inblandade från uppdragsgivarens sida, handledare och examinator. Här bestämdes hur arbetet skulle utföras och vilka programspråk som skulle användas. Programspråken valdes delvis av uppdragsgivaren då de hade vissa specifika krav på vilken utvecklingsmiljö som fick användas. Webbapplikationen var tvungen att utvecklas i ASP.NET eftersom alla program inom företaget ska använda samma ramverk och av säkerhetsskäl. Eftersom Visual Studio stödjer ASP.NET-utveckling i flera programmeringsspråk lämnades det valfritt. Här valdes C# eftersom det är väldigt likt Java. Stora delar av designen av webbsidan bestämdes direkt då handledaren på Peab ansåg att det är bättre för kunderna att webbsidan, där ordern läggs, är så lik den tidigare analysorderblanketten som möjligt för att inte försvåra inlärningsprocessen av det nya sättet att lägga ordrar.

3.2 Programmering

Programmeringen av prototypprogrammet gjordes i programspråket Java. Valet att använda Java var inget krav utan det valdes på grund av att där besatts störst kunskap och eftersom det tidigare i utbildningen har utvecklats en applikation i Java där kommunikation med en SQL-databas sker genom JDBC och det därmed fanns en färdig mall på hur det kan se ut. Eftersom prototypprogrammet inte kommer att sättas i bruk utan endast skapades för att kunna visa hur webbapplikationen kan användas tillsammans med det laboratorieprogram som de använder nu hade Peab inte några krav på hur detta program skulle utvecklas.

3.3 Databas och webbapplikation

Databasdesign skedde i samband med webbdesignen då ASP.NET har vissa inbyggda funktioner för användarautentisering och registrering vilket bara är möjligt genom att använda en databas. Därför skapades alla databaser i Visual Studio. För att förstå hur Visual Studio och ASP.NET fungerade användes tutorials på Youtube och även olika forum. Visual Studio innehåller ett flertal funktioner för att underlätta programmeringen av webbapplikationer bland annat för användarregistrering, inloggning och lösenordshantering. Med dessa funktioner saltas och hashas lösenorden automatiskt. Dessa funktioner hittades först efter att en egen metod för detta hade skapats.

Webbapplikationen är till stor del kodad i C#. Den logik som inte är kodad i C# är kodad i JavaScript. Dessa script krävdes för att för att hindra felinmatning genom att inaktivera textboxar och för att visa kalendrar vid sökning mellan datum.

3.4 Arbetssätt

Alla program har skrivits på egen laptop då datorerna som fanns på campus inte hade de program/tillägg som krävdes. Detta medförde att bara en kan arbeta med projektet hemma, därför schemalades träffar på campus för att ständigt ligga i fas och kunna jobba tillsammans. Då det inte är lätt för två att skriva programkod samtidigt har den ena kodat medan den andra sökt efter lösningar på fel som uppstått eller skrivit rapport. För att hitta lösningar har felmeddelanden googlats och de flesta problem har lösts genom sökning på forum så som stackoverflow, w3schools och Microsofts egen hemsida för metoder och klasser msdn.microsoft. Då det inte alltid har gått att få svar har "trial and error" varit den enda lösningen vilket i sin tur var väldigt tidskrävande.

En stor del av arbetet har utförts på Campus Helsingborg då ingen utrustning från Peab har krävts och en i gruppen redan var väl förtrogen med betonglaboratoriumets arbetsprocesser. Besök har gjorts på Peab med jämna mellanrum för att de skulle få en möjlighet att se vad som skapats och kunna påverka design och funktionalitet på webbsidan. Då prototypprogrammet skulle vara så likt programmet som redan används som möjligt togs bilder av det nuvarande programmets interface för att sedan kunna bygga en prototyp som är så lik denna som möjligt. Kontakten med andra inblandade företag har till stor del skett via telefon och mail men möten har även hållits på Peab för att lättare kunna visa och beskriva tanken bakom val som gjorts och arbete som utförts, och även för att få återkoppling och bekräftelse.

Programmeringen var en iterativ process där problemen ofta bröts ner till mindre och ofta förenklade delproblem som istället kunde lösas. Efterhand som förståelsen byggdes upp utnyttjades sedan kunskapen för att skapa en lösning till grundproblemet.

När arbetet startade saknades större kunskap om programmering av webbapplikationer, därmed blev mycket av koden som var skriven i början av projektet ändrad för att få en mer stabil webbsida. Till stora delar gjordes hela designen för ordersidan om i slutskedet för att få ett snyggare och mer stabilt interface efter att större kunskaper om HTML och CSS hade förvärvats.

3.4 Felsökning

För att hitta brister i webbapplikationen lades en mängd ordrar där tänkbara inmatningar vid orderläggning simulerades. Därefter reflekterades över tydligheten då t.ex. en kund försöker lägga en order och ordern inte kunde läggas antingen på grund av att en order med samma märkning och fabriksnummer redan fanns eller för att obligatoriska fält inte hade fyllts i.

Felsökning och test av webbsidan gjordes manuellt genom att försöka överbelasta och stresstesta sidan. Metoderna som användes var bland annat att anropa flera olika funktioner samtidigt och även genom att manuellt anropa en funktion flera gånger i en hög hastighet.

3.5 Källkritik

De källor som använts har i stor del varit ett forum och en blogg. På forumet publicerar man sina fel/problem och får svar från andra medlemmar på forumet. Svaren som givits är inte någon säker lösning då det kan dra ner på säkerheten i programmet eller skapa nya problem, det behöver inte ens vara så att det löser problemet. Fördelen med forum är att de tillhandahåller olika alternativa lösningar och synpunkter från andra användare. Dessa synpunkter kan därefter verifieras hos mer tillförlitliga källor. På *stackoverflow.com* var det enkelt att se vilket svar som löst flest personers problem då det finns ett röstningssystem på hur många som kunnat lösa sitt problem med hjälp av just det svaret. Detta forum har över 7 miljoner användare som hjälps åt att lösa varandras problem i programmering [4]. Bloggen som använts är en programmeringsblogg där bloggaren har publicerat handledningar om hur man kan implementera olika ovanliga funktioner. Det enklaste sättet för att verifiera var att testa att lägga in liknande metoder i koden och att utnyttja kunskap från tidigare kurser för att reflektera över eventuella säkerhetsbrister.

För att ta reda på information om hur Visual Studio fungerar har handledning via Youtube använts. Youtube är en välkänd sida för så kallade tutorials och även fast inte allt är anpassat för svenska versioner av programmet går mycket att förstå ändå. Detta användes mest i inledningsskedet för att få en introduktion till utvecklingsmiljön. Den bristande tillförlitligheten hos mediet har därmed inte haft negativ inverkan på slutresultatet.

msdn.microsoft[9] är Microsofts egna webbssida. Denna sida innehåller dokumentation över de olika metoder och funktioner som finns inbyggda i Microsofts program. Denna källa är dock svårtolkad då de oftast inte ger konkreta exempel på hur en metod kan användas. Eftersom det är Microsofts egenutvecklade sida som är till för att hjälpa deras användare med Microsofts produkter är detta en säker källa.

Kurslitteratur från två tidigare lästa kurser, Säkerhet EDA625 och Databasteknik EDAF20, användes. När en bok används som kurslitteratur bör det vara en säker källa som innehåller bra information. Under kursernas gång hittades små fel i böckerna men när man redan hade kunskap om dessa fel så blir källan säkrare. Litteraturen som användes för Säkerhetskursen [2], är svår att tolka och upplägget av information saknade en röd tråd. I kursen Databasteknik användes [3] vilket var en bra bas för inlärning av databashantering. Projektet *Krusty Cookies* som gjordes i kursen användes som grund till prototypprogrammet som utvecklades. Denna grund kunde hjälpa till att få en bra och säker struktur på programmet.

C# 3.0 programmering är en populär handbok för utveckling av webbapplikationer i C# som användes för att försäkra att lösningarna är *best practice* och således inte introducerar säkerhetshål. Eftersom författaren anställdes av Microsoft efter att ha skrivit en mängd böcker om deras produkter C# och ASP.NET är det mycket troligt att litteraturen är tillförlitlig.

Då det var väldigt svårt att hitta historik och allmän information om XML användes Wikipedia [7]. Eftersom vem som helst kan ändra på informationen Wikipedia är det ingen säker källa. Det finns länkar till var informationen är hämtad men stor del av länkarna var till andra mindre trovärdiga sidor såsom bloggar. Därför nämns inte mer information om XML i kapitel 2.1.

4. Analys

Strukturen för webbsidan byggdes med målet att vara så enkel och användarvänlig som möjligt men ändå innehålla säkerheten och funktionerna som var nödvändiga. Prototypprogrammet skulle endast göras i demonstrationssyfte och och därmed inte innehålla alla de funktioner som det riktiga labbprogrammet från MMP innehåller. Databasen byggdes enligt de metoder som lärdes ut i tidigare kurs i databasteknik med bland annat ER-diagram[3].

4.1 Uppbyggnad av webbapplikationen

För att få ett så enkelt och användarvänligt interface som möjligt används ett navigationsfält i toppen då det är väldigt vanligt förekommande på webbsidor. Detta fält innehåller olika länkar beroende på om användaren är inloggad eller inte. När användaren ej loggat in ser fältet ut enligt *figur 4.1*.

Figur 4.1 navigationsfält.

De olika rubrikerna länkar till olika sidor. Färgen orange används då det är en av färgerna på Peabs logga och är även en tydlig färg. Information och kontaktsidan lämnades ganska tomma då det är något Peab asfalt och Peab IT tillsammans ska stå för.

När en användare är inloggad kan den lägga en order. Order sidan är uppbyggd för att vara så lik Word-dokumentet som tidigare användes för lägga ordrar som möjligt. Den är även anpassad för labbprogrammet från MMP som används. Sidan anpassades genom att använda *drop down lists* vilket tvingar användaren till att använda ett objekt som redan finns i laborationsprogrammet. Om objektet inte finns i programmet finns skrivfältet *övrigt* där man kan be tekniker på laboratoriet lägga till objektet.

Tidigare har ordar lagts med både ett frysprov och ett tryckprov då de ofta testar en betongblandning för dessa samtidigt men i laborationsprogrammet kan bara en typ av provning registreras i taget. Detta har inneburit att labratorieteknikerna på Peab har varit tvungna att alltid ta en kopia på analysordern med de två olika typer av provning för att sedan manuellt göra om det till två olika prover. Detta problem löstes genom att en order bara kan läggas på en typ av provning i taget men för att inte beställaren ska behöva skriva om en exakt kopia av ordern sparas tidigare ifylld information tills användaren klickar på länken *ny order* igen eller lämnar sidan. För att undvika förvirring meddelas beställaren att ordern har lagts och vid vilken tidpunkt det skedde.

Uppbyggnaden av sidan för att titta på tidigare lagda ordrar var ett svårt moment där det vägdes mellan design, funktion och användarvänlighet. För att användaren enkelt skulle kunna söka efter en lagd rapport på ett specifikt datumintervall på ett lätt sätt utan att veta hur databasen läser datum implementerades en kalenderfunktion. Denna fungerar som en drop down list fast med en kalender istället. Eftersom det skulle kunna söka inom ett intervall av datum krävdes två sådana funktioner, en för början och en för slutet av intervallet. När användaren valt datum var webbsidan tvungen att göra en pageload för att databasen skulle

kunna registrera valen. Att den gör en PostBack för sidan när ett val är gjort i kalendrarna fungerar bra förutom om det är så att kunden vill söka på en annan månad. När man flyttade en månad fram eller tillbaka gjorde sidan en ny pageload vilket gjorde att användaren var tvungen att öppna upp kalendern igen fast nu stod den på en ny månad. Detta problem löstes genom ett event för *visible month changed*. När ett sådant event registreras ska servern anropa en metod som visar kalendern igen efter att sidan har laddats om.

För att det inte skulle finnas några obligatoriska sökfält användes en sökfunktion i databasen med *like* och ett "%" för att markera att resten kan vara vad som helst. Det innebär att databasen tar det som användaren matat in och tar fram allt som innehåller denna inmatning, till exempel om användaren söker på ett order id 3 får användaren fram 3, 30, 31, 32...300 osv. men om användaren lämnar skrivfältet tomt kommer alla order id visas.

När användaren sökt efter en en lagd order skulle den visas på något vis och gå att hämta hem som ett word dokument. För detta användes till en början listbox. I *listboxen* visades order id, märkning, datum och orderstatus. För att sedan hämta ordern behövde bara användaren markera ordern och sedan klicka på *Hämta*-knappen. Denna typen av lista fungerade bra förutom när det kom till design. Det blev otydligt vad som var order id eller märkning då allt stod på samma rad utan några kolumnuppdelningar eller spalter. Därför var det svårt för en användare som inte visste exakt all information om rapporten att hitta rätt. *Listboxen* byttes därmed ut mot *listview* som fungerade mer som en riktig tabell där allt var uppdelat i ett rutnät vilket gjorde det väldigt enkelt att se vad det var för resultat man fick på sin sökning. Problemet med *listview* var att det fanns inget användarvänligt sätt att bygga upp interfacen för att hämta den eftersökta rapporten. Valet föll därför tillbaka på *listbox*, men istället för att visa en hel rad med information om ordern kan användaren istället välja vilket av attributen som ska visas i listan och kan därmed försäkra sig om att det är rätt order de hittat. Som förinställt värde att visa används order id då det är värdet laboratoriet använder sig av och det blir då lättare för beställare och laboratorieteknikern att kommunicera om en specifik order.

När en order hämtas från databasen till prototypprogrammet byter ordern status från *Väntar på behandling* till *Under behandling*. Denna funktion används för att beställaren som lagt en order enkelt ska kunna se om laboratoriet har fått ordern och börjat behandlingen på den. För att funktionen ska fungera krävs en koppling mellan databas och program. Om denna funktion inte känns nödvändig skulle man istället kunna använda en XML-fil för att hämta in informationen från databasen.

Även fast ordersidan ser ut att vara enkelt uppbyggd så krävdes det väldigt mycket kod för att få varje textfält att kommunicera med databasen. Först döptes varje textbox var för sig för att lättare kunna hålla reda på vilken textbox som var vilken. När en textbox skapas blir det förinställda namnet på den *textbox#*. Efter att namnet ändrats skulle varje box begränsas till ett max antal tecken som fick skrivas in. För att veta hur stort maxantalet skulle vara testades det hur många tecken som fick plats i det dokument som ordern hamnar på efter den är lagd. Det som gjorde det ännu mer komplicerat var att ta reda på vad som en användare kunde få för sig att skriva in för typ av variabler i en textruta då man måste tilldela varje objekt i databasen ett max värde och vad det var för typ av värden som skulle gå att skriva in.

För att beställaren inte ska behöva fylla i information om sig själv på varje order sparas alltid den information som användes i den senaste lagda ordern av just den användaren. Detta är möjligt genom att använda ett attribut av typen *datetime* i databasen som anger när ordern lades. På så vis kan man se vilken beställarinformation som senast användes.

4.2 Säkerhet

Att finna alla säkerhetsbrister var svårt. Som utgångspunkt används informationen från säkerhetskursen där det lades stor vikt vid hur viktigt det var att hasha och salta och att ha starka lösenord[2]. För att alla användare ska få starka lösenord har fyra krav sats: Lösenordet måste vara åtta tecken eller längre, det måste även minst innehålla en bokstav, en siffra och ett specialtecken. Dessa krav ställs med hjälp av *RegularExpressionValidators* som finns i Visual Studio. För salt och hashning, som tidigare nämnts, fanns redan en inbyggd funktion för detta vilket gjorde att fokus hamnade på nästa säkerhetsåtgärd: skydd mot bottar.

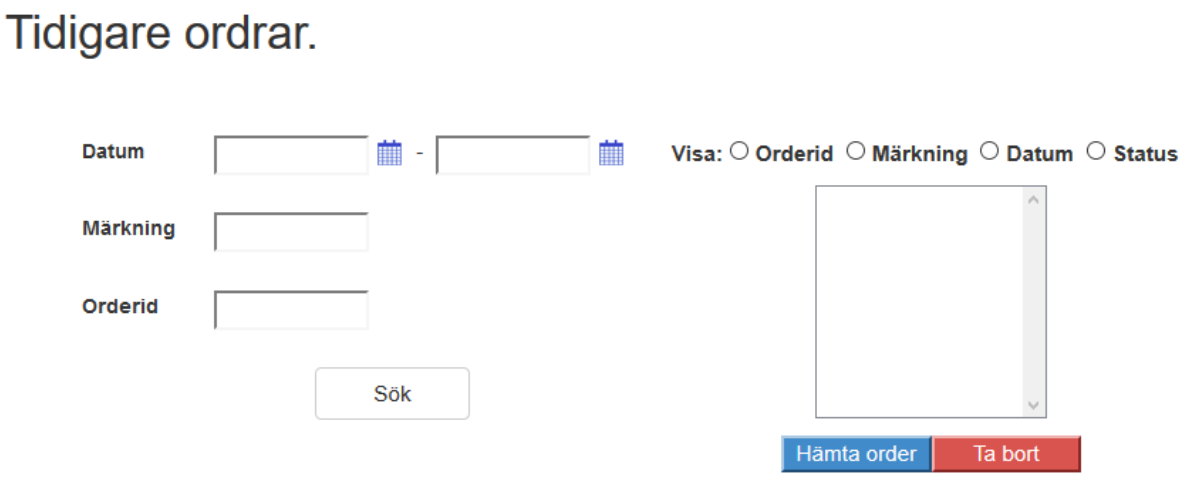
För att inte någon skulle kunna skapa ett flertal konto för att sedan överbelasta databasen lades en funktion för e-mejl konfirmation när ett konto har skapats. Denna funktion blir aldrig använd i det program som lämnades över till Peab asfalt då det kräver en separat mejl-service vilket inte kändes rätt att lägga pengar på då Peab IT redan äger en sådan service och kan då själva implementera denna om de vill. Denna e-mejl funktion kan även användas då någon glömt sitt lösenord. För att lägga en order måste användaren ha skapat ett konto vilket innebär att de som hanterar databasen enkelt kan se om någons aktiviteter skulle se ovanliga ut.

Tillgången till webbsidan begränsas för användare som inte har loggat in. Sådana anonyma användare skickas automatiskt vidare till inloggningssidan när de försöker komma åt en sida som kräver inloggning. Alla sidor som ligger i en mapp benämnd *Protected* är begränsade till inloggade användare.

Eftersom webbapplikationen både ska användas av personal på laboratoriet och personal i fabrikerna (beställarna) krävdes en implementation av roller. Laboratoriepersonalen tilldelas rollen som *admin* och fabrikspersonal tilldelas rollen som *user*. Denna rollimplementation var något som krävdes för att inte alla skulle ha tillgång till samma sidor. En admin har, förutom tillgång till alla sidor som en user har, också tillgång till en sida för att se alla kunders tidigare ordrar. En *admin* kan bara skapas i direkt kod för att inte någon ska kunna utge sig att vara en del av laboratoriet och därmed få tillgång till alla inskickade ordrar. Alla som registrerar sig på webbsidan blir en *user*. Rättigheterna mellan rollerna skiljer sig i sidan för att se lagda ordrar. För admin ser sidan ut enligt *figur 4.2*.

figur 4.2 Admins version för att hantera lagda ordrar.

Här kan man som *admin* välja att enbart visa de ordrar som ej är hämtade till prototypprogrammet genom att välja checkboxen *Endast obehandlade*. Så fort en order är inlagd i prototypprogrammet ändras status på den hämtade analysordern från obehandlad till behandlad. Denna funktion implementerades för att man som beställare ska se att laboratoriet har mottagit och påbörjat analysen och även för laboratoriet själva för att lättare ha koll på vilka ordrar som är nyinkomna. *Admin* kan även söka efter en specifik beställares ordrar. De två ovan nämnda funktionerna kan bara nås från *admin* rollen för att det är bara laboratoriet som ska ha tillgång till alla lagda ordrar medan en *user* bara ska ha tillgång till sina egna och behöver därmed inte ha dessa funktioner se *figur 4.3*.



Figur 4.3 Användarens version för hantering av tidigare lagda ordrar.

En *user* kan välja att radera en lagd order under hela den dag som ordern är lagd. Denna funktion används för att låta den som lagt ordern ha en chans på sig att göra om ordern om det skulle ha blivit fel vid inmatningen. Därför kan inte laboratoriet hämta ordern förrän dagen efter ordern är lagd. För att man inte av misstag ska råka radera en lagd order används en pop-up ruta som frågar om man är säker på att man vill ta bort den markerade filen.

För att implementera denna varningsruta användes en kontrollfunktion från AJAX Control Toolkit v17.1.

I databasen kan man inte se vem som är *admin* eller *user* då denna information är hashad.

De olika sidorna i webbapplikationen är placerade i mappar för vilka användaråtkomsten regleras. Det innebär att bara vissa har tillgång till de sidor som ligger i en specifik mapp. Till exempel kan sidan *administrera ordrar* bara nås om man är inloggad som admin även om användaren skriver in rätt länk för just den sidan i adressfältet medan alla har tillgång till andra sidor även om man inte är inloggad.

För att få en säker anslutning mellan användare och server används SSL/TLS, vilket dels autentiserar servern så att användaren vet att den är den som den utger sig för att vara, och dels krypterar kommunikationen så att den inte kan avlyssnas av utomstående. För att göra detta krävs ett certifikat genom vilket serverns identitet fastställs och nyckelutbyte sker. När programmet överlämnades användes ett självsignerat certifikat och tanken var att Peab IT ska ansöka om ett certifikat verifierat av en *Certificate Authority*.

4.3 Funktioner och Metoder

4.3.1 Läsa recept från XML

Eftersom all information om betongreceptet redan finns i producentens program *SIMMA* ska en exportfunktion där betongreceptet kan exporteras som XML-fil införas. Därför implementerades en metod för att ladda in ett recept från en XML-fil i formuläret (*figur 4.4*). Denna XML-fil läggs på beställarens dator och kan sedan med hjälp av en *Filväljare* väljas vid orderläggning. När filen har valts kan beställaren trycka på en knapp för att hämta receptet. Samtliga värden som fyllts i *SIMMA* kopieras till motsvarande textrutor i receptdelen av formuläret.

```
if (FileUpload1.HasFile)
{
    using (MemoryStream stream = new MemoryStream(FileUpload1.FileBytes))
    {
        //Skapar ett nytt XML-dokument och laddar in från vald receptfil.
        XmlDocument doc = new XmlDocument();
        doc.Load(stream);

        //Väljer alla undernoder till recept-noden i xml-dokumentet och itererar över dessa.
        XmlNodeList nodeList = doc.GetElementsByTagName("recept");
        foreach (XmlNode node in nodeList)
        {
            gjutdatum.Text = node["gjutdatum"].InnerText;
            märkning.Text = node["märkning"].InnerText;
            betonglev.Text = node["leverantör"].InnerText;
            avseddalder.Text = node["ålder"].InnerText;
            ...
```

Figur 4.4 Importfunktion för XML.

Eftersom XML-filen innehållande receptet är relativt liten kan hela filen läsas in i minnet. Därför används en *MemoryStream* istället för *FileStream* då filen läses in. Eftersom filen endast används för att fylla i textrutorna ger det fördelen att temporära filer inte behöver lagras och snabbare behandling eftersom inget behöver skrivas till disk [4][12].

4.3.2 Om order inte kunde läggas

För att det tydligt ska framgå om en order kunde läggas sätts *MaintainScrollPositionOnPostback* till *true* då sidan laddas, vilket innebär att när knappen för att lägga ordern har klickats på återgår browsern till samma position efter att en postback har gjorts. Under knappen ges ett meddelande som säger om ordern lades eller om något problem uppstod som gjorde att den inte kunde läggas.

4.3.3 Kontroll innan order läggs för att undvika dubbletter

För att kolla om märkning och fabriksnummer redan finns i samma rad innan en ny rad skapas då ordern läggs finns det i MySQL en *on duplicate key*-funktion. Denna funktion saknas dock i Microsoft SQL så detta löstes istället genom att först göra en query som kollar om en rad med detta fabriksnummer och märkning finns och sedan utförs en kontroll “if @@rowcount = 0 Begin...”. Den kod som följer efter *begin* utförs då endast om det inte redan fanns en rad i tabellen.

4.3.4 Validering av inmatad objektstorlek

Problem uppstod då det skulle ställas krav på att ett och endast ett av de tre fälten för storlek på provobjektet (kub, cylinder, borrkärna) ska fyllas i. Första tillvägagångssättet var att sätta *RequiredFieldValidator* på alla tre textrutorna och att låta all logik utföras på serversidan med *AutoPostBack* på textrutorna. Detta blev en klumpig lösning eftersom det då krävs att användaren klickar utanför boxen för att en postback ska göras, varefter de andra boxarna med tillhörande validerare inaktiveras. Detta fick också till följd att användaren inte direkt kan klicka i en kryssruta eftersom den postback som sker då textboxen lämnas gör att den ändringen inte registreras.

Nästa lösning på problemet var att med JavaScript låta logiken utföras på klientsidan. Boxarna kan då inaktiveras så fort en knapptryckning registreras i textrutan (*figur 4.5*) utan att postbacks behöver skickas till servern. Processen blev på så vis snabbare och mer responsiv. För att validera att ett av fälten har fyllts i användes sedan en extra textruta, dold för användaren, som fylls i med samma innehåll som de andra så fort något skrivs i ett av dessa (*figur 4.6*). En *RequiredFieldValidator* användes sedan på denna textruta.

```
<asp:TextBox Style="margin-left: 10px" ID="kubstorlek" ClientIDMode="Static" runat="server" Width="40px" onkeyup="kubDisable()"
```

Figur 4.5 Bindning av kubDisable-funktionen till keyup-eventet.

```
function kubDisable() {
    $('#hiddenbox').val($('#kubstorlek').val());
    if ($('#kubstorlek').val() !== '') {
        $('#borrkärnestorlek').prop('disabled', true);
        $('#cylinderstorlek').prop('disabled', true);
    }
    else {
        $('#borrkärnestorlek').prop('disabled', false);
        $('#cylinderstorlek').prop('disabled', false);
    }
}
```

Figur 4.6 Implementering av textboxinaktivering då kubstorlek anges.

Motsvarande funktioner används för de andra två textrutorna.

4.3.5 Skapa rapport

När en order läggs och en rapport ska skapas ska namnet på filen innehålla det orderid som ordern har fått. Eftersom detta orderid inte sätts manuellt utan alltid inkrementeras med ett, så uppstod problemet att avgöra vad aktuellt orderid var. Detta löstes genom att en databas-query körs som tar reda på det högsta orderid i databasen efter att ordern har lagts. Detta orderid hämtas därefter ut ur det *ResultSet* som returneras och injiceras därefter i sökvägen för den fil som ska skapas.

När beställaren lägger en order ska en rapport skapas som sedan kan skrivas ut och sparas. Eftersom laboratoriet i dagsläget skriver ut och sparar det orderformulär som beställaren fyllt i låg det nära till hands att använda samma formulär som mall och att fylla det med information från databasen då rapporten ska skapas.

Eftersom Word-dokument är uppbyggda som ZIP-filer innehållande XML-dokument kan taggar skapas för att åstadkomma detta. I Words utvecklarläge kunde innehållskontroller sättas in på de platser data ska injiceras och i dessa kontrollers egenskaper kunde de namnges. Sedan användes ett annat program *Word Content Control Toolkit* för att skapa XML-taggar som eftersom kontrollen hade namngetts kunde kopplas ihop med denna. I webbapplikationen skapas en kopia av formuläret och sedan öppnas denna kopia. En ny XML-del skapas som en textsträng med taggar som matchar de som tidigare lagts till i *Word Content Control Toolkit.* Därefter tas tidigare *CustomXMLPart* bort ur dokumentet och ersätts med den nya innehållande den data som ska infogas. Med en *stream writer* skrivs sedan den nya XML-koden in i CustomXML-delen av Word-dokumentet (*figur 4.7*) [11].

```
using (WordprocessingDocument doc = WordprocessingDocument.Open(FilledFormPath, true))
{
    //Skapa XML-del som matchar taggarna i CustomXML
    string newXML = "<root>" +
        "<Orderid>" + reader["orderid"].ToString() + "</Orderid>" +
        "<Antal>" + reader["antal"].ToString() + "</Antal>" +
        ...
        "<Original>" + reader["Orgtill"].ToString() + "</Original>" +
        "<Kopia>" + reader["Kopiatill"].ToString() + "</Kopia> " +
        "</root>";

    //Ta bort tidigare CustomXMLPart
    MainDocumentPart mainDoc = doc.MainDocumentPart;
    mainDoc.DeleteParts<CustomXmlPart>(mainDoc.CustomXmlParts);

    //Lägg till den nya CustomXMLPart innehållande aktuell data
    CustomXmlPart customXml = mainDoc.AddCustomXmlPart(CustomXmlPartType.CustomXml);
    using (StreamWriter sw = new StreamWriter(customXml.GetStream()))
    {
        sw.Write(newXML);
    }

    doc.Close();
}
```

Figur 4.7 kod för överföring från XML-dokument.

4.3.6 Åtkomsträttigheter för olika användare och roller

För att styra vilka roller som har tillgång till vilken sida och hur *navigation bar* ska presenteras för olika roller måste först en *RoleProvider* konfigureras i web.config (*figur 4.8*) [9]. Eftersom rollerna lagras i tabeller i SQL-databasen används SQL som RoleProvider.

```
<roleManager defaultProvider="SQL" enabled="true" cacheRolesInCookie="true">
  <providers>
    <clear/>
    <add connectionStringName="TestingConnectionString" applicationName="/" name="SQL" type="System.Web.Security.SqlRoleProvider"/>
  </providers>
```

Figur 4.8 SQL som RoleProvider.

När RoleProvider hade satts i web.config i roten kunde åtkomsträttigheterna för sidor placerade i de olika mapparna konfigureras i den web.config-fil som ligger under Account-mappen. Här nekas åtkomst för alla sidor placerade i *Protected*-mappen för alla anonyma användare (“?”) och alla övriga (“*”) tillåts åtkomst.

På samma vis styrs åtkomst till alla sidor i *Admin*-mappen, fast denna gång baserat på vilken roll användaren har (*figur 4.9*). Här tillåts endast de användare som har rollen *Administration* åtkomst. Övriga användare omdirigeras till inloggningssidan då de försöker öppna sidan.

```
<?xml version="1.0"?>
<configuration>

  <location path="protected">
    <system.web>
      <authorization>
        <deny users="?"/>
        <allow users ="*"/>
      </authorization>
    </system.web>
  </location>

  <location path="protected/admin">
    <system.web>
      <authorization>
        <allow roles="Administration"/>
        <deny users = "*"/>
      </authorization>
    </system.web>
  </location>

</configuration>
```

Figur 4.9 Åtkomstkontroll.

5 Resultat

Det slutliga resultatet av uppbyggnaden av hela systemet består av fem olika delar (enligt fig 5.1).

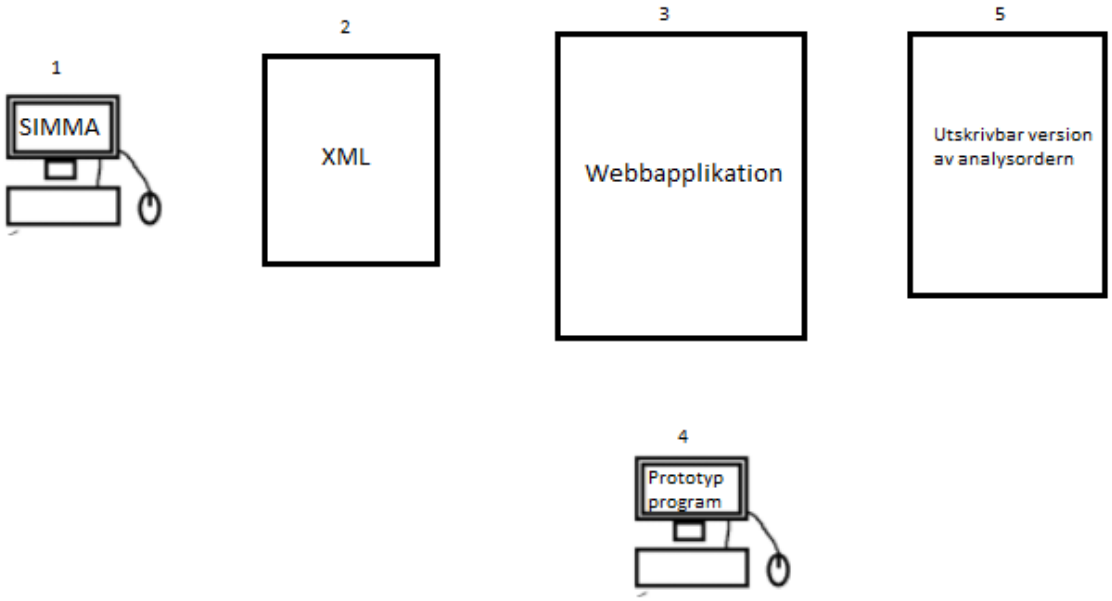


Fig 5.1 *De fem delarna av systemet.*

Del 1 är programmet som beställarna på Swerock använder sig av. Det är i detta program all information om betongen matas in för blandning. Denna information extraheras sedan ut till ett XML-dokument (2). Genom det extraherade dokumentet kan nu beställaren lägga en order utan att själv behöva mata in all information manuellt. Genom att istället importera XML-dokumentet till webbapplikationen (3) och med hjälp av dess taggar fylls alla fält i receptdelen av formuläret automatiskt. Dygnet efter att beställaren lagt en order kan nu prototypprogrammet hämta den lagda ordern genom att bara fylla i det *order id* som ordern fick vid registreringen. Alla analysordrar som ännu ej är behandlade i prototypprogrammet kan ses i ett fält inuti programmet. När en analysorder skapas i webbapplikationen skapas ett utskrivbart Word-dokument (5) som är baserat på den analysorderblankett som tidigare användes, vilket sedan kan nås av den som skapade ordern och även av laboratorieteknikern.

5.1 Registrering

I fliken *Registrera* kan användaren skapa ett konto. Det som krävs är en giltig e-postadress och ett lösenord på minst åtta tecken varav minst ett är bokstav, siffra samt specialtecken. Hur innehållet på denna sida ser ut ses i fig 5.2.

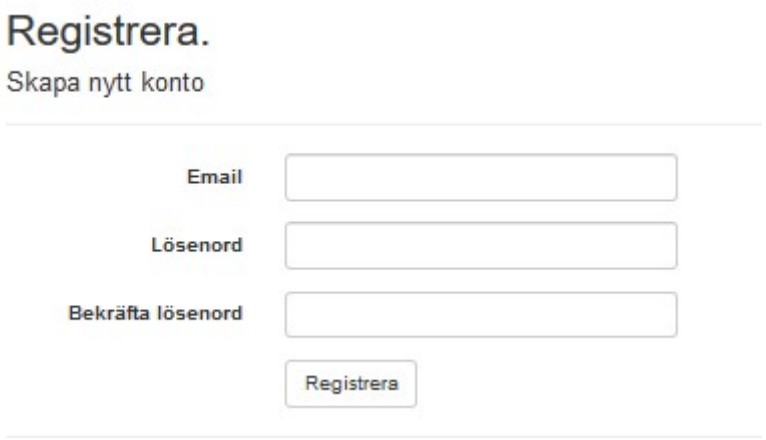


Fig 5.2. Registreringssida.

Om något fält inte är korrekt ifyllt får användaren direkt en varning.
Har användaren redan ett konto väljer den fliken *Logga in*.

5.2 Logga in

Fig 5.2. Inloggningssida.

I fliken *Logga in (se fig 5.2)* kan användaren välja att logga in, registrera sig eller om den glömt lösenord kan den välja att den får ett nytt lösenord genererat och skickas till användarens mail. Funktionen för glömt lösenord fanns ej när sidan överlämnades till Peab då de vill använda sina egna mejlservice tjänster.

När användaren är inloggad ändras navigationsfältet i toppen av sidan.

Före inloggning:

Efter inloggning:

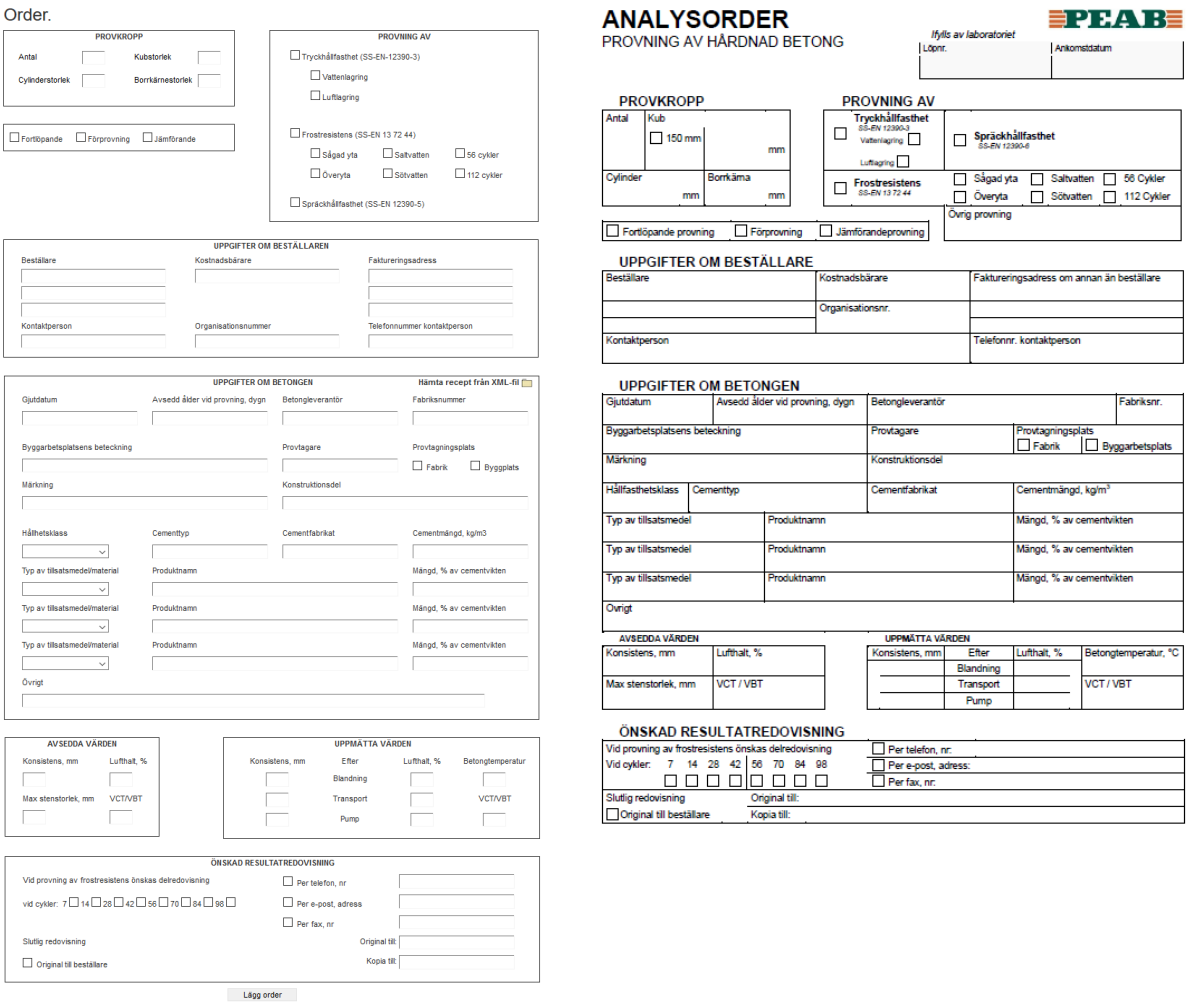
Fig 5.3. Navigationsfält före och efter inloggning.

Användaren har som inloggad möjligheten att lägga en ny order, titta på tidigare lagda ordrar och ändra lösenordet på kontot den är inloggad med. Länken som visar användarnamnet för den inloggade är en *drop down list* men är även en länk till en sida genom vilken samma funktionalitet uppnås. Den har dubbla funktioner för att det ska gå lika bra att navigera på sidan genom att använda mus som att bara ha ett tangentbord till hands.

5.3 Ny order

Huvudfunktionen i applikationen var att en användare skulle kunna lägga en order som sedan kunde hämtas utan manuell inmatning av hela analysordern utav laboratorieteknikern. För att få denna funktion att fungera användes en databas som gav varje lagd order ett unikt ID. Detta ID kunde sedan hämtas genom den sammankoppling mellan databas och prototypprogrammet som fanns.

Det som efterfrågades från Peab var en ordersida som var så lik den mall som då användes för att lägga ordrar som möjligt för att underlätta inlärningen av det nya ordersystemet. Nedan visas en jämförelse mellan ordersidan som skapats och Word-dokumentet som använts tidigare (*figur 5.4*). För tydligare illustrationer se Appendix.



Figur 5.4 Applikationens orderflik i jämförelse med analysorderdokumentet som tidigare använts.

På ordersidan finns det en knapp som låter användaren med hjälp av en XML-fil ladda upp hela ordern. Varför just denna dokumenttyp används var för att det inte skulle finnas någon koppling mellan databasen som skapats och Swerocks datorprogram *SIMMA*. Detta var ett krav från Swerock. Istället kan *SIMMA* exportera en XML-fil som sedan kan läsas in i orderformuläret och därmed försvinner dubbelarbetet som pågår för de som tillverkar betongen.

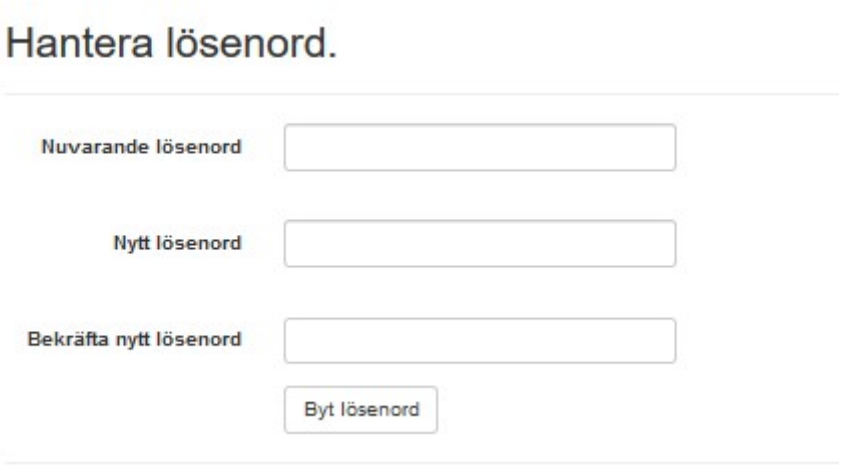
5.4 Lagda ordrar

I dropdown menyn, på användarnamnet, kan *Tidigare ordrar* väljas. Denna länkar användaren vidare till en sida där de kan titta på sina tidigare lagda ordrar. På denna sida går det att söka efter en rapport med olika separata eller kombinerade attribut: märkning, datum och id. Om det skulle uppstå något problem angående en order kan laboratoriet och beställare med hjälp av dessa funktioner lätt gemensamt hitta den eftersökta ordern. Som tidigare visats och beskrivits har rollerna *admin* och *user* olika rättigheter och versioner av denna sidan.

När en order hämtas hämtar databasen det Word-dokument som skapats när ordern tidigare lagts. Detta dokument är detsamma som tidigare använts av beställare när de lagt en analysorder. Att ha en funktion för att skapa en utskrivbar version av lagda ordrar var också ett krav från peab då de behöver förvara en kopia av den i ett brandsäkert skåp.

5.5 Ändra lösenord

Den andra länken i dropdown-menyn är *Ändra lösenord*. Användaren länkas då vidare till en sida där de kan byta det gamla lösenordet till ett nytt (se *figur 5.5)*. Kraven på det nya lösenordet är detsamma som när ett konto skapas. För att användaren ska vara säker på att det ändrar till rätt nytt lösenord krävs det att de återupprepar lösenordet en gång. Det nya lösenordet hashas och saltas precis som tidigare.



Figur 5.5 hantera lösenord.

6 Slutsats

De flesta frågor/problem gick att lösas och det slutliga resultatet uppfyller de funktioner som efterfrågats. Då erfarenhet inom Visual Studio saknades när arbetet började tog det längre tid än väntat att uppnå det önskade resultatet.

Kan hemsidan som ska skapas göras tillräckligt säker och uppfyller webbsidan Peab IT:s krav angående säkerhet och utvecklingsmiljö?

Då Peab IT har ett antal specifika krav angående säkerhet och uppbyggnad av program var det inte möjligt att uppfylla alla dessa utan de på Peab IT fick själv lägga på extra kod. Hemsidan uppfyller dock flertalet säkerhetskrav som kan ställas på en vanlig webbsida. Dessa krav uppfylls t.ex. genom hashning av all information om användaren, genom att inte lagra konfidentiell eller för mycket personlig information i databasen samt skydd mot SQL-injections med hjälp av prepared statements. Begränsning av antalet tillåtna tecken i alla textrutor bidrar också till ökad säkerhet.

Hur ska webbinterfacet utformas för att vara så användarvänligt som möjligt?

Webinterfacet utformades på ett lättigenkännligt sätt genom att använda ett enkelt navigationsfält på toppen av sidan med länkar som är tydliga om vart användaren leds. Färgerna är klara och tydliga. Genom att konstruera ordersidan på ett sådant sätt att den är så lik det tidigare order dokumentet som användes och sätta krav på de delar som behövs för labbet blir det lätt för både nya och gamla beställare.

Hur ska vi implementera en funktion för extrahering av data från databasen till det nya labbprogrammet som ska skapas?

För extrahering av data från databasen till labbprogrammet sker kommunikationen direkt genom JDBC. Labbprogrammet skickar en query till databasen för att få ut all data om raden som har det orderid som eftersöks. Attributen plockas därefter ut ur resultatet och lagras som textsträngar i en lista som sedan returneras till den klass som representerar orderfliken från vilken metoden anropades. Ett efter ett plockas attributen ut ur listan och fylls i motsvarande textfält.

Vilka programmeringsspråk ska användas?

Visual Studio med ASP.NET var krav på att det skulle användas vilket var bra då det var ett enkelt program att göra webbapplikationer med. I Visual Studio valdes C# som språk då det programmeringsspråket är väldigt likt Java. Java användes för att skapa prototypprogrammet då gruppmedlemmarna hade kunskap om kommunikation mellan databaser och Java från tidigare lästa kurser.

Hur ska vår prototyp integreras i de befintliga programmen?

Prototypen som skapades har ingenting med labbprogrammet som används att göra. MMP kan välja att använda delar av koden till prototypprogrammet till att lägga in i deras program för att snabbare implementera de *queries* som utförs mellan databasen och programmet.

Kan vi lägga in en funktion i programmet som med hjälp streckkodsläsare kan skanna in utskrivna ordrar?

Hade tiden räckt till hade en funktion för att generera streckkod på orderrapporten skapats, men då flera företag kom med i bilden och ingen i gruppen hade tidigare erfarenhet av att skapa webbapplikationer räckte inte tiden till. Eftersom analysordrarna nu oftast kommer komma digitalt så finns det heller ingen tidsvinst av att använda sig av en streckkodsläsare.

Hur ska databasen utformas för att kunna samarbeta med både en webbsida och ett program?

Det fanns inget särskilt som behövde tas i beaktning vid designen av databasen för att samarbetet skulle fungera.

6.1 Reflektion över etiska aspekter

För att undvika att lagra känslig information i databasen har webbapplikationen begränsats till att bara själva analysordrarna ska läggas där och inte resultaten. Detta förhindrar att konfidentiell information läcker ut om applikationen skulle bli utsatt för hot. Resultaten från laboratoriet skickas via mejl till beställarens angivna adress och därmed blir det färre mellanhänder som kan läcka ut den sekretessbelagda informationen.

När en användare registrerar sig behöver den inte ange några personliga uppgifter mer än användarens mejladress som blir saltad och hashad direkt efter registreringen. Även rollen som användaren får blir hashad vilket innebär att det blir väldigt svårt för någon annan att ta reda på vem användaren är och vad den har tillgång till.

Ikonen för kalendern som används på sidan för att titta på tidigare lagda ordrar och den för att hämta från XML-fil på orderläggningssidan har skapats av examensarbetarna i programmet Paint. Detta gjordes för att undvika copyright-intrång.

Då Peab inte får skapa ett program som liknar eller kan ersätta det labbprogram som används saknar prototypprogrammet många av de viktigaste funktionerna för att det skulle vara ett fungerande labbprogram. Inga bilder visas på labbprogrammet som används av säkerhetsskäl.

6.2 Framtida utvecklingsmöjligheter

Det finns många utvecklingsmöjligheter för webbsidan. Dels har Peab IT en del att lägga till för att få den precis som de vill enligt deras standarder, dels kan det även finnas en efterfrågan från andra liknande företag såsom NCC och Skanska att ha en egen version av webbapplikationen. Den applikation som är utvecklad är just nu konstruerad just för Peab men med bara lite justering kan den anpassas för andra liknande företag. Applikationen hade även kunnat användas för asfaltsdelen på Peab då inmatning av information i programmet fungerar ungefär på samma sätt där som för betong.

Det hade även gått att göra applikationen ännu säkrare genom att begränsa vilka domäner som kommer in på sidan. Till exempel kunde registreringen begränsas till användare som använder sig av mejladresser som innehåller en av Peabs eller dess dotterbolags mejldomäner. Detta hade gjort att någon som inte har någonting med Peab att göra inte kunde lägga någon order. Det finns även andra lösningar till detta såsom att den som vill lägga en order på webbsidan först måste skicka ett mejl till Peab för att be om att få ett konto. Då går det att verifiera att användaren verkligen har med betonganalyser att göra.

6.3 Peabs kommentarer och tankar

Peab är väldigt nöjda med den skapade applikationen men det kommer inte att driftsättas än då alla inblandade parter måste godkänna och koda om deras program innan det kan börja fungera på riktigt. Tiden som de beräknade skulle sparas i bara dubbelarbetet var totalt sju minuter per order. Sammanlagt beräknar de att de skulle spara en hel månads tid i arbete då alla inkomna analysordrar under ett år lades ihop. Detta inkluderar inte den tid som besparas då de felinmatningar, som nu elimineras, måste hanteras. Peab IT kommenterade att de ofta måste skriva om hela programmet om det är gjort av någon utomstående för att det ska uppfylla deras krav till punkt och pricka men efter möte med ansvariga på Peab IT var de nöjda med både uppbyggnad och resultat. MMP kände att de hade en del arbete att göra på labbprogrammet som används men var klart intresserad av att skapa en koppling mellan deras program och webbapplikationen som skapats.

7 Terminologi

1. *PostBack* är när en sida skickas tillbaka till servern för processering efter att någon ändring skett på klientsidan
2. *Best practice* innebär den metod som passar bäst att använda vid just de tillfället som är bättre än alla andra funktioner just då.
3. *PageLoad* laddar om hela sidan för att uppdatera datan.
4. *RegularExpressionValidators* ASP.NET-kontroll som kollar så att den inmatade datan matchar kravet som ställs för texten som kan skrivas där.
5. *RequiredFieldValidator* ASP.NET-kontroll som gör att ett skrivfält blir obligatoriskt att fylla i.
6. *Certificate Authority* är någon som verifierar servern och ger ut ett certifikat som verifierar dess autenticitet.

8 Källförteckning

1. *Liberty J, Xie D. C# 3.0 programmering. 5 uppl. Sundbyberg: Pagina Förlags AB; 2008. s 329-331.*
2. *Gollman, Dieter: Computer Security, John Wiley & Sons, 2010, ISBN: 9780470741153 s50-63*
3. *Padron-McCarthy T, Risch T. Databasteknik. Upplaga 1:13. Lund: Studentlitteratur; 2014. s27-53.*
4. *ForEach Loop XmlNodeList, https://stackoverflow.com/questions/11847965/foreach-loop-xmlnodelist, senast hämtad 2017-05-23.*
5. *Javascript https://www.java.com/sv/download/faq/java_javascript.xml, senast hämtad 2017-05-12.*
6. *Javascript* https://developer.mozilla.org/sv-SE/docs/Web/JavaScript*, senast hämtad 2017-05-12*
7. *XML https://en.wikipedia.org/wiki/XML senast hämtad 2017-05-19*
8. *C# introduction, https://docs.microsoft.com/en-us/dotnet/articles/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework senast hämtad 2017-05-22*
9. *Providers element for RoleManager, https://msdn.microsoft.com/en-us/library/ms164661(v=vs.100).aspx, senast hämtad 2017-04-20.*
10. *Ajax Introduction, https://www.w3schools.com/xml/ajax_intro.asp, senast hämtad 2017-05-21*
11. *Populating Word 2007 Templates through OpenXML, https://seroter.wordpress.com/2009/12/23/populating-word-2007-templates-through-open-xml/, senast hämtad 2017-05-23.*
12. *Reading XML-file, https://stackoverflow.com/questions/2679963/reading-the-xml-file-in-server-without-saving-it, senast hämtad 2017-05-23.*

9A. Figurer och bilder

ANALYSORDER

PROVNING AV HÅRDNAD Bbetong

PEAB

Ifylls av laboratoriet

Löpnr.	Ankomstdatum

PROVKROPP

Antal	Kub ☐ 150 mm	mm
Cylinder mm	Borrkärna mm	

PROVNING AV

☐ **Tryckhållfasthet** *SS-EN 12390-3* Vattenlagring ☐ Luftlagring ☐	☐ **Spräckhållfasthet** *SS-EN 12390-6*
☐ **Frostresistens** *SS-EN 13 72 44*	☐ Sågad yta ☐ Saltvatten ☐ 56 Cykler ☐ Överyta ☐ Sötvatten ☐ 112 Cykler
☐ Fortlöpande provning ☐ Förprovning ☐ Jämförandeprovning	Övrig provning

UPPGIFTER OM BESTÄLLARE

Beställare	Kostnadsbärare	Faktureringsadress om annan än beställare
	Organisationsnr.	
Kontaktperson		Telefonnr. kontaktperson

UPPGIFTER OM BETONGEN

Gjutdatum	Avsedd ålder vid provning, dygn	Betongleverantör	Fabriksnr.
Byggarbetsplatsens beteckning		Provtagare	Provtagningsplats ☐ Fabrik ☐ Byggarbetsplats
Märkning		Konstruktionsdel	
Hållfasthetsklass	Cementtyp	Cementfabrikat	Cementmängd, kg/m³
Typ av tillsatsmedel	Produktnamn		Mängd, % av cementvikten
Typ av tillsatsmedel	Produktnamn		Mängd, % av cementvikten
Typ av tillsatsmedel	Produktnamn		Mängd, % av cementvikten
Övrigt			

AVSEDDA VÄRDEN

Konsistens, mm	Lufthalt, %
Max stenstorlek, mm	VCT / VBT

UPPMÄTTA VÄRDEN

Konsistens, mm	Efter	Lufthalt, %	Betongtemperatur, °C
	Blandning		
	Transport		VCT / VBT
	Pump		

ÖNSKAD RESULTATREDOVISNING

Vid provning av frostresistens önskas delredovisning Vid cykler: 7 ☐ 14 ☐ 28 ☐ 42 ☐ 56 ☐ 70 ☐ 84 ☐ 98 ☐	☐ Per telefon, nr: ☐ Per e-post, adress: ☐ Per fax, nr:
Slutlig redovisning ☐ Original till beställare	Original till: Kopia till:

Order.

PROVKROPP

Antal | Kubstorlek

Cylinderstorlek | Borrkärnestorlek

☐ Fortlöpande ☐ Förprovning ☐ Jämförande

PROVNING AV

☐ Tryckhållfasthet (SS-EN-12390-3)
- ☐ Vattenlagring
- ☐ Luftlagring

☐ Frostresistens (SS-EN 13 72 44)
- ☐ Sågad yta ☐ Saltvatten ☐ 56 cykler
- ☐ Överyta ☐ Sötvatten ☐ 112 cykler

☐ Spräckhållfasthet (SS-EN 12390-5)

UPPGIFTER OM BESTÄLLAREN

Beställare | Kostnadsbärare | Faktureringsadress

Kontaktperson | Organisationsnummer | Telefonnummer kontaktperson

UPPGIFTER OM BETONGEN — Hämta recept från XML-fil

Gjutdatum | Avsedd ålder vid provning, dygn | Betongleverantör | Fabriksnummer

Byggarbetsplatsens beteckning | Provtagare | Provtagningsplats ☐ Fabrik ☐ Byggplats

Märkning | Konstruktionsdel

Hållhetsklass | Cementtyp | Cementfabrikat | Cementmängd, kg/m3

Typ av tillsatsmedel/material | Produktnamn | Mängd, % av cementvikten

Typ av tillsatsmedel/material | Produktnamn | Mängd, % av cementvikten

Typ av tillsatsmedel/material | Produktnamn | Mängd, % av cementvikten

Övrigt

AVSEDDA VÄRDEN

Konsistens, mm | Lufthalt, %

Max stenstorlek, mm | VCT/VBT

UPPMÄTTA VÄRDEN

Konsistens, mm	Efter	Lufthalt, %	Betongtemperatur
	Blandning		
	Transport		VCT/VBT
	Pump		

ÖNSKAD RESULTATREDOVISNING

Vid provning av frostresistens önskas delredovisning

vid cykler: 7 ☐ 14 ☐ 28 ☐ 42 ☐ 56 ☐ 70 ☐ 84 ☐ 98 ☐

☐ Per telefon, nr

☐ Per e-post, adress

☐ Per fax, nr

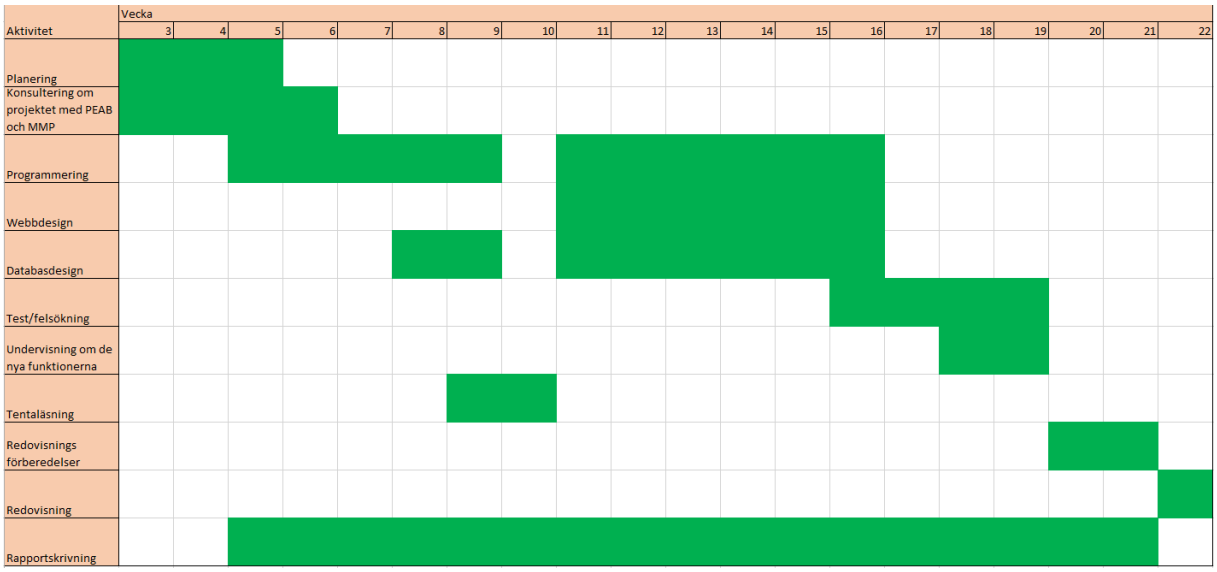
Slutlig redovisning

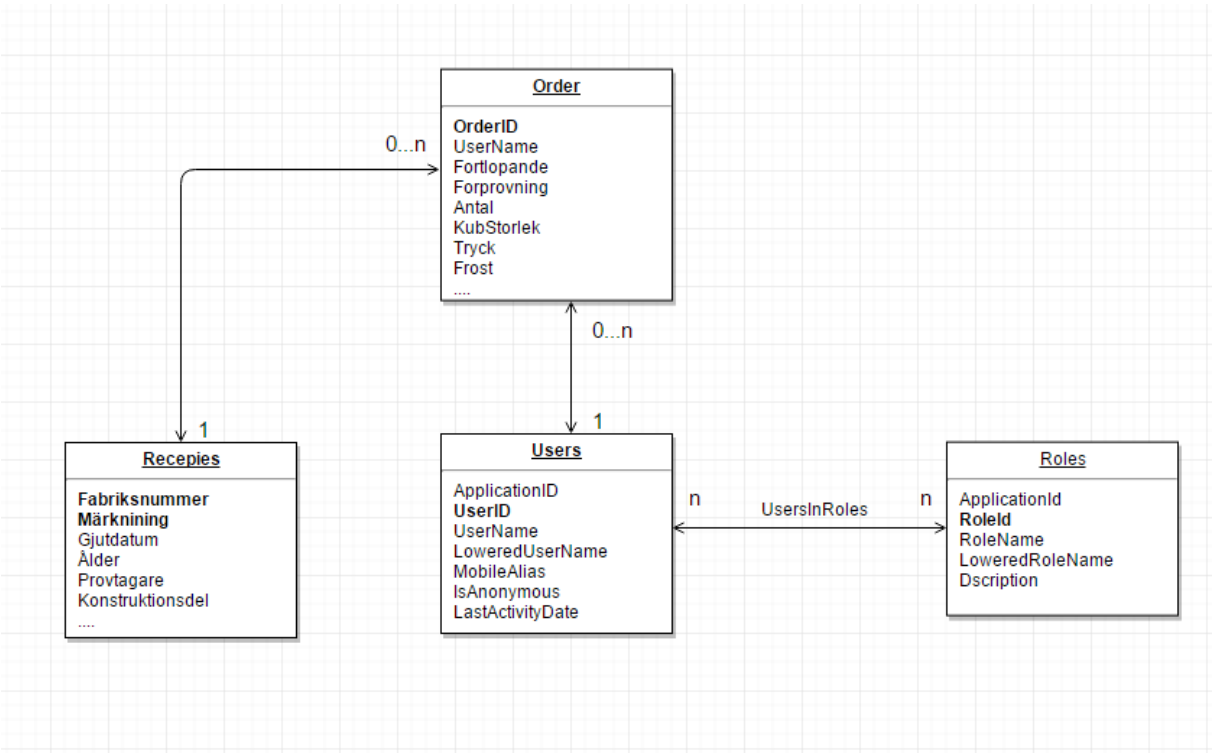
☐ Original till beställare

Original till:

Kopia till:

Lägg order


Aktivitet
Vecka
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
Planering
Konsultering om projektet med PEAB och MMP
Programmering
Webbdesign
Databasdesign
Test/felsökning
Undervisning om de nya funktionerna
Tentaläsning
Redovisnings förberedelser
Redovisning
Rapportskrivning


Order
OrderID
UserName
Fortlopande
Forprovning
Antal
KubStorlek
Tryck
Frost
....
0...n
0...n
1
1
Recepies
Fabriksnummer
Märknining
Gjutdatum
Ålder
Provtagare
Konstruktionsdel
....
Users
ApplicationID
UserID
UserName
LoweredUserName
MobileAlias
IsAnonymous
LastActivityDate
n
UsersInRoles
n
Roles
ApplicationId
RoleId
RoleName
LoweredRoleName
Dscription