YRKESHÖGSKOLAN NOVIA

Webbapplikation för kartläggning av skadliga ämnen och labbanalys

Conny Nyman

Examensarbete för Tradenom (YH)-examen

Utbildningsprogrammet i Informationsbehandling

Raseborg 2018

EXAMENSARBETE

Författare: Conny Nyman
Utbildning och ort: Informationsbehandling, Raseborg
Handledare: Klaus Hansen

Titel: Webbapplikation för kartläggning av skadliga ämnen och labbanalys

---

Datum 4.2.2018 Sidantal 23 Bilagor 1

Abstrakt

Examensarbetet handlar om utvecklingen av en webbapplikation, vars syfte är att underlätta och effektivera arbetet vid kartläggning av skadliga ämnen. Webbapplikationen gör det möjligt för kartläggare att skapa och färdigställa kartläggningsprojekt. Ett projekt innehåller information så som bottenplan, prover, analyssvar och mängdförteckning. Projektdata läggs till i olika skeden av kartläggningsprocessen. Slutligen kan kartläggaren generera en rapport över kartläggningen som skickas till kunden. En mobilapplikation har även utvecklats i samband med webbapplikationen. Mobilapplikationen tillåter kartläggare att mata in information om tagna prover ute på fältet.

Webbapplikationens frontend och backend är uppdelad i två olika ramverk. Backend hanteras av SilverStripe som är ett CMS system med ett PHP ramverk. Frontend hanteras av AngularJS som är ett JavaScript ramverk.

Implementering av ett REST gränssnitt har utvecklats för att möjliggöra kommunikation mellan mobilapplikationen, frontend och backend. REST-gränssnittet erbjuder CRUD funktionalitet och har utvecklats med hjälp av två SilverStripe moduler.

I arbetet beskrivs alla utvecklingsfaser av webbapplikationen, design, teknikval och implementering. Därtill beskrivs webbapplikationens funktion i praktiken och utmaningar som uppstod i samband med utvecklingen och respektive lösningar.

Språk: Svenska Nyckelord: SilverStripe, AngularJS, Webbapplikation, REST

BACHELOR'S THESIS

Author: Conny Nyman
Degree Programme: Business Information Technology, Raseborg
Supervisor: Klaus Hansen

Title: Web Application for Mapping of Harmful Substances and Lab Analysis

---

Date 4.2.2018 Number of pages 23 Appendices 1

Abstract

This thesis covers the development of a web application which purpose is to facilitate and streamline the work with mapping of harmful substances. The web application offers the possibility for mappers to create and complete mapping projects. A project contains information such as floor image, samples, analyses and a variety list. Project data is inserted at different stages of the mapping process. The mapper can generate reports for finished projects, which then can be sent to the customers. A mobile application was also developed alongside the web application. The mobile application enables the mappers to insert data regarding samples out on the field.

The web application's frontend and backend are separated in two different frameworks. Backend is handled by SilverStripe which is a CMS system with a PHP framework. Frontend is handled by AngularJS which is a JavaScript framework.

REST API enables communication between the mobile application, frontend and backend. The REST interface offers CRUD functionality and is developed with the help of two SilverStripe modules.

All development phases of the web application, design, choice of technologies and implementation are described. The web application's use in practice and challenges that arose during the development as well as the solutions, are also described.

Language: Swedish Key words: SilverStripe, AngularJS, Web application, REST

Innehållsförteckning

Figurförteckning

1. Inledning

Examensarbetet handlar om ett beställningsarbete på en webbapplikation. Uppdragsgivaren är ett företag som utför labbanalyser för asbest och andra farliga ämnen. Webbapplikationen har jag utvecklat hos ett systemutvecklingsföretag, som utvecklar mobila lösningar och fullständiga webblösningar. Projektet startade genom labbföretagets idé om att förbättra arbetssättet för hur beställningar och utföranden av analyser sker. Detta blev möjligt genom att erbjuda ett webbsystem för kartläggare, som förenklar arbetet vid kartläggningar. Webbsystemet som jag har utvecklat består av en webbapplikation med stöd för extern kommunikation med mobila applikationer genom ett Representational State Transfer (REST) gränssnitt.

1.1 Syfte

Syftet med examensarbetet är att beskriva hur webbapplikationen har utvecklats samt hur kommunikationen fungerar mellan webbapplikationen och mobilapplikationen. Webbapplikationens syfte är att strukturera och förenkla processen för hur det går till då en kartläggare beställer en analys, till att rapporten är färdigställd. Mobilapplikationens syfte är att förenkla arbetet ute på fältet genom att spara provexemplar till webbapplikationen samtidigt som kartläggaren är på plats och tar fysiska provbitar.

2. Teknik

I detta kapitel beskrivs den teknik som använts under utvecklingen. Teknikuppsättningen för utveckling av webbapplikationen består främst av SilverStripe, AngularJS, Gulp och Bower. Valet av teknikerna grundar sig på systemutvecklingsföretagets långa erfarenhet. Systemutvecklingsföretaget har valt att använda SilverStripe framom andra Content Management Systems (CMS) eftersom det har ett flexibelt PHP ramverk och en mångfunktionell CMS modul. Företagets uppsättning av tekniker har visat sig vara stabilt under flera års tid i olika projekt. I och med att jag använde en testad och bekant teknikuppsättning, fanns möjligheten att få ta del av värdefulla råd från arbetskollegor när problem uppstod under utvecklingen.

2.1 SilverStripe

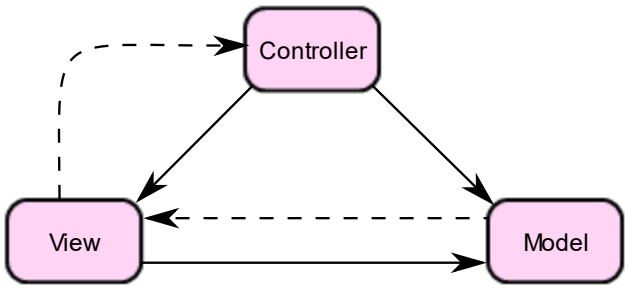
SilverStripe är ett CMS system med öppen programvara. I Innowebs (2018) artikel om SilverStripe står att systemet används världen över av bland annat regeringar, företag och ideella organisationer. SilverStripe erbjuder funktioner som krävs vid skapandet av webbsidor, intranät och webbapplikationer. SilverStripe är i grunden ett PHP webbapplikationsramverk. Ramverket är flexibelt och innehåller grundläggande principer såsom säkerhetsmodeller, arbetsflöde och caching. CMS modulen består av ett användargränssnitt som tillåter inloggade användare i systemet att hantera innehåll på webbsidan. Modulen erbjuder möjlighet att begränsa behörigheter för användare.

Achilles (2018) lyfter följande fördelar med SilverStripe. SilverStripe är en öppen programvara med BSD-licens. Det är flexibelt att använda och har en välskriven dokumentation. Det har även en normaliserad databasstruktur och ett intuitivt administrativt verktyg där man kan finjustera användarroller och behörigheter. Nackdelar med att använda SilverStripe menar Uzayr (2013) är att det finns ett litet utbud av teman. CMS systemet kräver även goda kunskaper i kodning.

Magnusson (2010) skriver att SilverStripe CMS blev certifierad av Microsoft. Det var det första system med öppen källkod att bli detta.

2.1.1 Model-View-Controller

SilverStripe baserar sig på arkitekturmönstret Model-View-Controller, som illustreras i figur 1. Arkitekturmönstret går ut på att separera data och affärslogik från interaktion i presentationsvyer, vilket möjliggör modifikation av data utan att påverka presentationsvyn. Det är Controllern som på basis av inkommande förfrågningar, bestämmer vilken View som visas samt meddelar till Model om data som skall hämtas, skapas eller modifieras i databasen. (Deacon 2005).



Figur 1. Model-View-Controller arkitekturen.

Model är en annan benämning för domänlager. Databasschemat skapas utgående från domänlagret. I SilverStripe utformas domänlagret på basen av subklasser av DataObject. För varje DataObject skapas en tabell i databasen och för varje egenskap skapas en kolumn i tabellen. I subklasser definieras även relationer mellan DataObject. Relationer gör det möjligt att länka ihop objekt från olika tabeller. I figur 2 framgår exempel på detta. I SilverStripe migreras ändringar i domänlagret genom att köra kommandot basURL/dev/build i webbläsaren. (SilverStripe u.å).

```
private static $db = [
    self::STATUS => 'Varchar(255)',
];
```

```
private static $has_many = [
    self::PROJECTS => Project::class,
];
```

Figur 2. Egenskap Status med typ Varchar definieras. Has_many relation till projekt definieras.

View representerar ett gränssnitt för domänlagret. Gränssnittet kan vara i olika former, t.ex. grafiskt gränssnitt, kommandotolk eller applikationsgränssnitt. I SilverStripe är View ett grafiskt gränssnitt som består av mallar. En mall kan hantera alla märkspråk, t.ex. HTML, CSV, JSON. I mallarna kan variabler läggas till med prefixet $. Variablerna fungerar som platshållare för data som kommer från Controller eller DataModel.

Controller innehåller skräddarsydd logik som används vid inkommande förfrågningar från användarinteraktionen. När Controllern har behandlat informationen returneras respons till klienten. I SilverStripe skapas skräddarsydda Controllers genom att ärva Controller klassen. En Controller i SilverStripe behöver en egen rutt definierad i routes.yml för att ta emot förfrågningar. Exempel på en rutt framgår i figur 3. Det är vanligt med en-till-en relation mellan URL strängen och respektive klass/metod -> exempel.fi/klass/funktion/id/

```
Director:
  rules:
    'authentication//$Action': 'AuthenticationController'
```

Figur 3. Rutt i routes.yml.

2.1.2 Object-Relational Mapper

Object-Relational Mapper (ORM) är en teknik som används för att hantera data i databasen på ett objektorienterat sätt. Genom att använda ORM kan man skriva förfrågningar till databasen i det språk man programmerar i. Fördelar med att använda ORM är bl.a. att det erbjuder metoder för att skapa och modifiera data, entiteterna cachas i minnet vilket minskar belastningen på databasen samt att flera användare kan modifiera samma data samtidigt.

SilverStripe har en egen ORM för att utföra olika förfrågningar till databasen. SilverStripe följer vissa regler för att ORM skall fungera, följande regler är tagna från (SilverStripe u.å) dokumentation:

- Varje databas tabell länkas till en PHP klass
- Varje rad i databastabellen länkas till ett PHP objekt
- Varje databaskolumn länkas till en egenskap på ett PHP objekt

Vid en respons där flera objekt ingår, returneras de som ett DataList objekt. Det finns olika metoder man kan köra på DataList objekt i SilverStripe. Alla metoder hittas under http://api.silverstripe.org/3/DataList.html.

SilverStripe ORM använder sig av Lazy loading design mönstret. Det bygger på att ingen databasförfrågan körs innan värdet från förfrågan används. ORM fortsätter att konstruera databasförfrågan ända tills värdet behövs, se figur 4. På detta sätt minskar antalet förfrågningar som skickas till databasen och effektiviteten i applikationen ökar.

```
$users = Member::get()->filter('FirstName', $firstName)
echo $users->Count();
// SQL - SELECT COUNT(*) FROM Member WHERE FirstName='Valfritt namn'
```

Figur 4. Lazy loading i SilverStripe ORM.

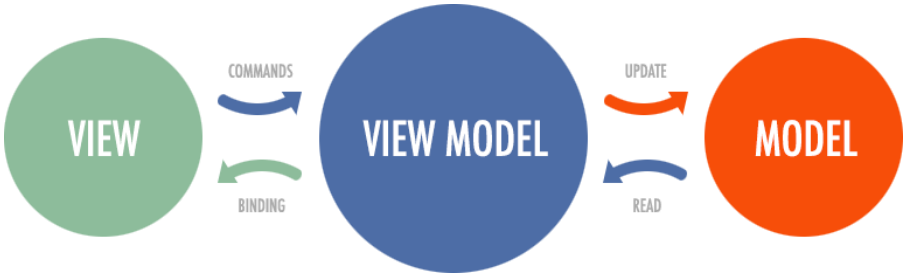
2.2 AngularJS

AngularJS är ett strukturellt ramverk för skapandet av webbapplikationer. Med AngularJS kan webbapplikationer av typen SPA Single-page application skapas. SPA innebär att webbapplikationen laddar in alla resurser endast en gång och behöver därefter inte laddas om. AngularJS grundar sig på det klassiska MVC designmönstret men har en egen variant MVVM, som står för Model, View, View-Model. I figur 5 illustreras MVVM.

- Model i AngularJS är data som kan hanteras i ViewModel och View. Model kan bestå av JavaScript variabler och objekt. Data som definieras i Controller/ViewModel utgör Model. Model i AngularJS är inte bunden till en databas, därav behövs egna lösningar implementeras för att spara förändringar. Vanligen består Model av dynamiska data i AngularJS. (Malik u.å)

- View i AngularJS definieras i märkspråket HTML. AngularJS utvidgar HTML syntaxen med ytterligare tag attribut. Attributen kallas Directives. Directives gör det möjligt att med View-Model binda ihop HTML input och output med Model definierade i Controller. Data i Model är alltid uppdaterat oavsett om förändringar utförs i View eller Controller. Detta är möjligt tack vare AngularJS tvåvägsdatabindning. Tvåvägsbindningen hanterar synkroniseringen mellan View och Model. (Malik u.å)

- View-Model i AngularJS är länken mellan View och Model. View-Model består av JavaScript funktioner definierade i en Controller. View-Model hanterar uppdateringen av Model och länkar värdet till View. View-Model behöver definieras både i Controller och View för att länken skall fungera. Länkningen möjliggörs genom användningen av Scope. Scope är ett objekt som innehåller referenser mellan View och Model. Enligt Ringo (2015) skall View-Model i AngularJS jämföras med en Model istället för en Controller i MVC arkitekturen.



Figur 5. Illustration av MVVM i AngularJS.

2.3 Bower

Bower är en pakethanterare som hanterar paket för frontend. Ett paket kan vara t.ex. ett bibliotek så som jQuery eller ett ramverk så som AngularJs. En manifestfil finns tillgänglig i Bower med namnet bower.json. I filen definieras alla paket som projektet är beroende av. Vanliga Bower kommandon är bower install och bower update. Kommandot bower install installerar alla paket som är definierade i manifestfilen. Kommandot bower update uppdaterar alla installerade paket enligt manifestfilen. Nya beroenden kan installeras och läggas till i manifestfilen genom att köra kommandot bower install tillsammans med <paket-namn> –save. Bower installerar även paketberoenden. Vid situationer där paket har samma beroenden så delar de på samma paket. Eftersom paketberoende delas, optimeras även laddningstiden i webbläsaren. Bower installerar paketberoenden endast en gång. Bower vägrar även att installera olika versioner av paketversioner. Orsaken är att det skulle förlänga laddningstiden. Bower underlättar arbetet vid förflyttning av projekt genom att installera alla paket med ett kommando. (Sapegin 2014).

2.4 Gulp.js

Gulp är ett webbspecifikt byggverktyg som används inom webbutveckling för att utföra definierade uppgifter. Gulp erbjuder möjligheten att automatisera repeterande uppgifter, t.ex. att kopiera filer till en vald mapp, uppdatera webbläsaren när en fil sparas eller kompilera Less/Sass till CSS. Gulp läser instruktioner från en konfigurationsfil kallad gulpfile.js. Därefter utför Gulp instruktionerna från filen. Funktionaliteten i Gulp kan utvidgas genom att installera olika typer av tillägg. Genom att låta Gulp hantera repeterande uppgifter ökas produktiviteten. (Clapp 2015).

3. Webbapplikationen

I detta kapitel presenteras webbapplikationens planering, design och uppbyggnad. I planeringen och designen beskrivs prototypen och enhetsdiagrammet för projektet. I uppbyggnaden av webbapplikationen beskrivs backend och frontend. Även kommunikationen mellan backend och frontend samt kommunikationen mellan backend och mobilapplikationen förklaras.

3.1 Utvecklingsmetod

En agil utvecklingsmetod användes för projektet. Genom detta kunde utvecklingen av applikationen startas omgående. En agil utvecklingsmetod innebär att utvecklaren kontinuerligt levererar nya funktioner eller uppdateringar i programvaran. Kontakten är tät mellan kunden och projektledaren. Därmed har kunden möjlighet att följa med utvecklingen och ge respons. En annan fördel är att delar av produkten kan tas i bruk förrän alla funktioner är implementerade. För att åstadkomma en agil utvecklingsmetod användes projektverktyget JIRA. JIRA erbjuder bl.a. backlogs, sprints, epics och tickets.

I praktiken innebar det regelbundna leveranser åt kunden och frekventa kundmöten för kommande funktioner. Funktioner som skulle implementeras lades till i JIRA som tickets enligt prioritetsordning. Prioritetsordningen skapades utgående från kundens önskemål. Varje vecka meddelade jag vilka tickets jag förväntade mig att få klara under veckan. Därmed hade kunden möjlighet att följa med hur projektet framskrider.

3.2 Planering och design

I detta kapitel behandlas planeringen av webbapplikationen. Planeringen omfattar möten mellan parterna, en prototyp och ett enhetssambandsdiagram. Planeringen av webbapplikationen utfördes i samråd mellan parterna. Planeringen genomfördes genom kontinuerliga kundbesök samt med en utvecklingsmiljö där kunden regelbundet kunde följa med processen. Kontakten var därmed tät mellan parterna och utvecklingen kunde ske snabbare. Webbapplikationens användningsfall planerades enligt följande:

1. Kartläggaren skapar ett nytt projekt med grundinformation i webbapplikationen.
2. Kartläggaren loggar in på mobilapplikationen och lägger till provexemplar till det aktuella projektet.
3. Labbanalytiker loggar in på webbapplikationen och fyller i analyssvar för proverna.
4. Kartläggaren loggar in på webbsystemet och fyller i åtgärder för analyssvaren.
5. Kartläggaren skriver ut en kartläggningsrapport via webbapplikationen.

3.2.1 Prototyp

I startprocessen producerades en webb-prototyp av systemutvecklingsföretagets designer, utgående från diskussionerna mellan parterna. Prototypen berättar till stor del både för utvecklaren och kunden hur slutprodukten skall se ut och fungera. En godkänd prototyp indikerar parterna har förstått produktens syfte. Under utvecklingen upptäcktes dock att systemet inte kan vara lika isolerat som prototypen ursprungligen var. I prototypen förekom heller inte egna vyer för kartläggare och labbanalytiker. Det implementerade jag i ett senare skede för att tydligare definiera vad en vy behandlar.

3.2.2 Enhetssambandsdiagram

Ett enhetssambandsdiagram utformades utifrån en prototyp, se bilaga 1. Ett enhetsdiagram ger en översikt av hur alla entiteter, som skall finnas i webbapplikationen, hänger ihop. Diagrammet innehåller information om hurdana tabeller och kolumner som skall finnas i systemet. Detta utgick jag ifrån vid utvecklingen av domänlagret för webbapplikationen.

3.3 Uppbyggnad

I detta kapitel beskrivs webbapplikationens uppbyggnad. Webbapplikationen är huvudsakligen utvecklad i två ramverk, SilverStripe och AngularJS. Backend hanteras av SilverStripe för hantering av logik och data. Frontent hanteras av AngularJS för visning av data i applikationens användargränssnitt. Förutom Angular används också Gulp.js och Bower i frontend. Gulp.js sköter om att packa och konvertera alla filer till en build mapp som webbläsaren kan läsa in filer från. Bower sköter hantering av paket som läggs till i index.html filen. Därmed är paketen tillgängliga i Angular projektet.

Utveckling med separata ramverk möjliggjordes genom systemutvecklingsföretagets SilverStripe modul som tillämpar ett REST gränssnitt (se avsnitt 3.3.3 för beskrivning av REST gränssnitt). Syftet med modulen är att öppna upp möjligheten för externa system att kommunicera med SilverStripe. Modulen erbjuder även en ObjectService klass med CRUD funktionalitet. CRUD står för create, read, update och delete. Genom att skapa subklasser av ObjectService kunde CRUD funktionalitet ärvas. Utvecklingen effektiverades därmed, eftersom jag inte behövde implementera CRUD funktionalitet enskilt för varje klass.

3.3.1 Backend

Backend står för den bakomliggande funktionaliteten i webbapplikationen som hanteras av SilverStripe. SilverStripe erbjuder en stabil grund för utveckling av webbapplikationer genom att hantera konton, behörigheter, DataObject med databasrelationer och dylikt. Detta medförde att arbetet effektiverades eftersom jag inte behövde utveckla själva grunden för applikationen. Jag kunde omedelbart börja utveckla produkten och presentera framsteg i applikationen för kunden. Första skedet i utvecklingsfasen gick ut på att definiera domänlagret. Databas tabeller och kolumner skapades utifrån den information som kunde tas ut från enhetsdiagrammet. Ett komplett domänlager ger en djupare förståelse för hur vyerna behöver utformas i frontend**.**

Systemets datadesign konstruerades enligt hur utvecklaren definierar domänlagret. För varje modell skapas en tabell i databasen bestående av kolumner där modellens egenskaper framstår. Ett noga genomtänkt domänlager stöder bättre förutsättningar för vidareutveckling. Utvecklingen av webbapplikationen påbörjades när domänlagret var definierat. I detta kapitel beskrivs de grundläggande delarna av domänlagret.

Webbapplikationen skulle erbjuda funktionalitet för både kartläggare och labbanalytiker. Detta innebar att jag var i behov av en modell med inloggningsmöjligheter och behörigheter. För att lösa detta använde jag mig av en existerande klass i SilverStripe, Member. Member klassen innehöll de egenskaper som rollerna krävde. Jag skapade DataObject Inspector och Analyst som ärver Member. Därefter lade jag till specifika egenskaper för respektive klasser.

Kartläggare som arbetar inom samma företag skulle ha möjlighet att se alla projekt tillgängliga i webbapplikationen. Lösningen blev ett DataObject Company. Company innehåller grundinformation om företag och relationer till Inspector. Således kunde filtrering baserat på företag implementeras.

Det skulle vara möjligt för både kartläggare och labbanalytiker att skapa projekt. Därav skapade jag Model Project som innehåller grundinformation om projekt. Projekt skapade av kartläggare skulle innehålla ytterligare följande information, bottenplan, provexemplar, provanalyser och provåtgärder. Projekt skapade av labbanalytiker skulle ytterligare innehålla provanalyser.

Projektet skulle innehålla information om bottenplan. För detta skapades Model Floor. Floor innehöll följande information, våningsnummer, våningstyp och bild på bottenplan. Eftersom projekten skulle ha en eller flera våningar, definierades en relation mellan modellerna. Bottenplan kunde läggas till från en vy skapad i Angular.

Det skulle vara möjligt att lägga till provexemplar till projekt via mobil applikationen. Detta löstes genom att skapa Model Sample. Sample innehåller grundläggande information om provexemplaret, bl.a. plats, material samt position X och Y. Position X och Y definierades eftersom det skulle gå att markera i mobilapplikationen exakt var provet hade tagits. Markeringen läggs sedan till i kartläggningsrapporten.

Labbanalytiker skulle ha möjlighet att lägga till provanalyser. Därav skapade jag Model Analysis. Analysis innehåller information om asbesttyper och andra skadliga ämnen bl.a. PAH, PCB, bly och tungmetaller. I laboratoriet utförs analyser på fysiska prover och analysresultat läggs till i webbapplikationen.

Det skulle vara möjligt för kartläggaren att lägga till åtgärdsförslag i webbapplikationen för prov som innehåller asbest eller andra skadliga ämnen. Åtgärdsförslag skulle läggas till i kartläggningsrapporten under mängdförteckningsdelen. Således har kunden möjlighet att se dem. Detta löstes genom implementering av Model ProposedAction. ProposedAction innehåller information om hurdana åtgärder ägaren till byggnaden där skadliga ämnen påvisats, uppmanas att göra.

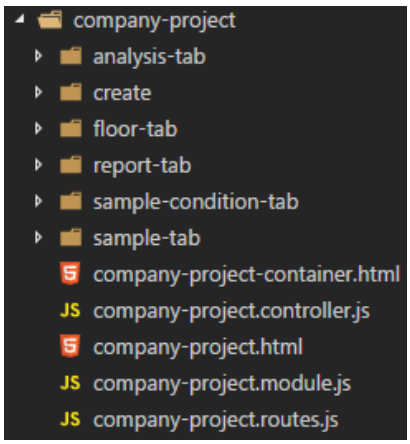
3.3.2 Frontend

Frontend står för den synliga delen av webbapplikationen och hanteras av AngularJS. AngularJS är ett mycket flexibelt ramverk som stöder utveckling av dynamiska vyer med användarvänliga grafiska gränssnitt. Företagets utvecklare har lång erfarenhet av ramverket. Därför valde jag att använda teknologin i detta projekt.

I AngularJS definieras vanligen en eller flera ViewModel/Controller för varje vy. I takt med att projektet växer, ökar även antalet filer. Därav är det viktigt att redan från start skapa en grund som stöder hållbar vidareutveckling. Utvecklingen av frontend påbörjades först då domänlagret var definierat. Därmed hade jag nödvändig information tillgänglig som krävdes för planeringen av vyerna och funktionaliteten för frontend.

Det grafiska gränssnittet utvecklades utifrån domänlagret och prototypen. Fokus låg på att gränssnittet skulle motsvara prototypen både grafiskt och funktionellt, eftersom det är vad kunden förväntar sig i den slutliga produkten. Det skapades vyer utöver de som var tillgängliga i prototypen. Detta eftersom vyerna i prototypen inte motsvarande alla entiteter i domänlagret.

Enligt Kukic (2018) stöder en logisk och väl genomtänkt mappstruktur god kod underhållbarhet. Detta kan uppnås genom att följa principer vid skapandet av filer och mappar. I detta projekt tillämpade jag följande principer, mappar namnges enligt vad vyn behandlar och komponenter bundna till samma vy läggs till i respektive mapp. Således hade jag en mapphierarki som är logiskt uttänkt och lätt att navigera i, se figur 6.



Figur 6. Logisk mapphierarki i AngularJS.

Vid utvecklingen av vyer var jag konsekvent. Detta för att samma intryck skulle bibehållas igenom hela webbapplikationen. Genom att vara konsekvent underlättades utvecklingen av vyer som ej förekom i prototypen. Att hålla sig till en konsekvent stil möjliggjordes genom användningen av ett UI-komponent ramverk AngularJS Material. Ramverket erbjuder bl.a. moderna grafiska komponenter med bra användarvänlighet som kan tillämpas i AngularJS projekt. Utöver grafiska komponenter erbjuds responsiv layout. Ruebbelke (2015) nämner

att ramverkets layout är baserat på CSS-flexbox. Med ramverkets responsiva design kan användarvänlighet som ger positiva upplevelser för användaren skapas. Det resulterar i att kunden vill fortsätta använda tjänsten.

Funktionaliteten för vyerna, så som hantering av händelser utifrån input, behandling av data och transporterande av data till backend möjliggjordes genom användningen av Controller. I version 1.2.0 av AngularJS implementerades syntaxen Controller As. Detta ändrade synen

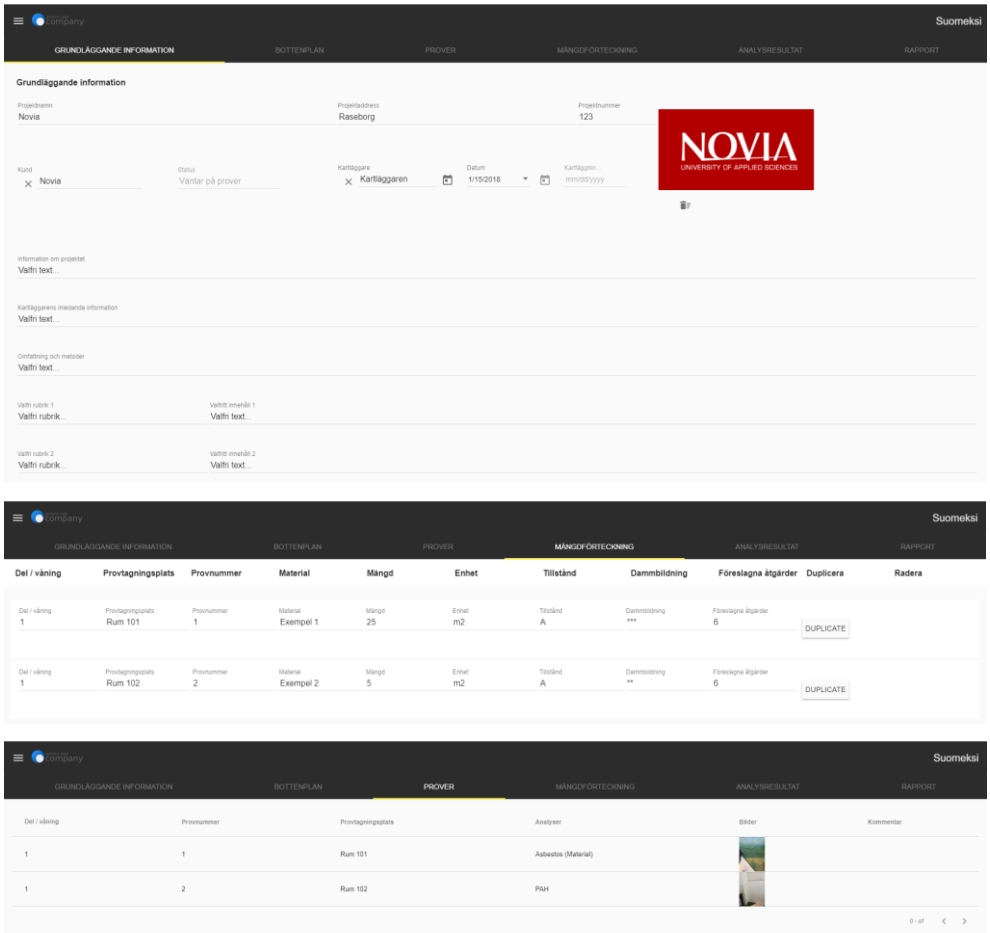
på hur en Controller kan användas för att skriva ut data till View. Det möjliggjorde att View kunde kommunicera med Controller utan att injicera Scope. I figur 7 framkommer kommunikationen mellan View och Controller utan Scope. Tidigare var det endast Scope objektet som möjliggjorde kommunikation mellan View och Controller.

```
<div ng-controller="ExamensarbeteCtrl as vm">
    <p>{{vm.info}}</p>
</div>
```

```
app.controller('ExamensarbeteCtrl', function() {
  var vm = this;
  vm.info = "kommunikation mellan View och Controller utan $scope";
});
```

Figur 7. Kommunikation mellan Controller och View utan Scope.

Principen en vy per arbetsprocess tillämpades vid utvecklingen av vyer och respektive komponenter. I figur 8 visas vyer där principen har tillämpats. Funktionaliteten i vyerna utformades med de olika arbetsprocesserna i fokus. Det resulterade i logiska vyer med genomtänkt funktionalitet. Det i sin tur leder till positiva upplevelser för användaren.



Figur 8. Vyer - grundläggande information, mängdförteckning och prover.

För färdiga projekt skulle kartläggaren ha möjlighet att generera en kartläggningsrapport. Kartläggningsrapporten skickas till slutkunden. Rapporten innehåller nödvändig information som är viktig för kunden. Stöd för rapportgenerering löstes med hjälp av ett JavaScript bibliotek vid namn PDFMake. PDFMake erbjuder möjlighet för PDF generering direkt i webbläsaren. Biblioteket har en kraftfull layout motor som erbjuder många användbara element bl.a. kolumner, tabeller, listor och bilder. Det går även att definiera egna stilar som kan återanvändas. En dokumentdefinition behöver skapas innan biblioteket kan skapa en PDF fil. I dokumentdefinitionen skall hela dokumentets struktur definieras. Exempel på dokumentdefinition framkommer i figur 9.

```
var dokumentDefinition = {
    content: [
        'Dokumentdefinition med kolumner',
        {
            columns: [
                {
                    width: 'auto',
                    text: 'Första kolumnen'
                },
                {
                    width: '*',
                    text: 'Andra kolumnen'
                }
            ]
        }
    ]
};
```

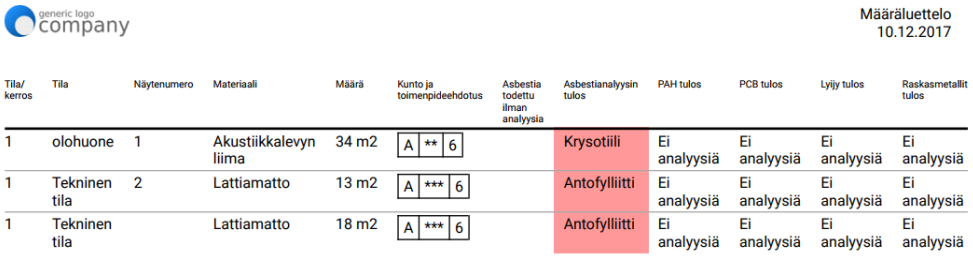
Figur 9. Dokumentdefinition för PDFMake.

Rapporten skulle innehålla mycket dynamiska data från databasen. Därav beslutade jag att skapa metoder som bearbetar dynamiska data och returnerar det i ett giltigt format för dokumentdefinition. Metoderna kunde återanvändas, vilket ökade produktiviteten. Exempel på en sådan metod är table metoden, se figur 10. Metoden tog emot data och kolumnnamn som parametrar och returnerade en tabell färdig att använda i dokumentdefinitionen.

```
function table(data, columns) {
    if(data != null && data.length > 0) {
        return {
            table: {
                headerRows: 1,
                body: buildTableBody(data, columns)
            },
            layout: 'lightHorizontalLines'
        };
    }
    return {};
}
```

Figur 10. Metoden table för skapandet av dynamiska tabeller.

Metoderna stöder bra kod underhåll och vidareutveckling av dokumentdefinitionen. En PDF fil kan sedan skapas genom användningen av metoden createPdf. Metoden tar in dokumentdefinitionen som parameter och returnerar en PDF fil. I figur 11 framkommer hur en tabell skapad i PDFMake kan se ut.


generic logo company

Määräluettelo
10.12.2017

Tila/ kerros	Tila	Näytenumero	Materiaali	Määrä	Kunto ja toimenpideehdotus	Asbestia todettu ilman analyysia	Asbestianalyysin tulos	PAH tulos	PCB tulos	Lyijy tulos	Raskasmetallit tulos
1	olohuone	1	Akustiikkalevyn liima	34 m2	A ** 6		Krysotiili	Ei analyysiä	Ei analyysiä	Ei analyysiä	Ei analyysiä
1	Tekninen tila	2	Lattiamatto	13 m2	A *** 6		Antofylliitti	Ei analyysiä	Ei analyysiä	Ei analyysiä	Ei analyysiä
1	Tekninen tila		Lattiamatto	18 m2	A *** 6		Antofylliitti	Ei analyysiä	Ei analyysiä	Ei analyysiä	Ei analyysiä

Figur 11. Tabell skapad i PDFMake.

3.3.3 Backend-frontend kommunikation med REST

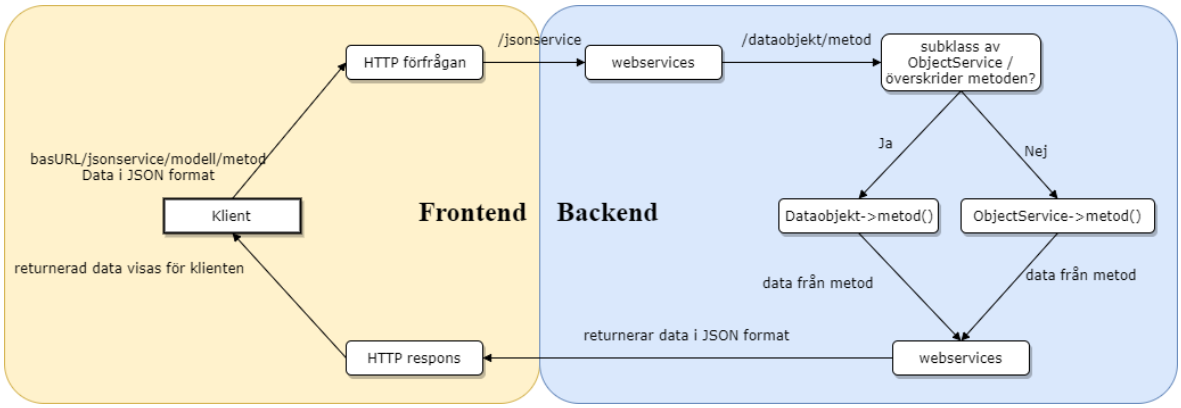
Kommunikationen mellan frontend och backend möjliggjordes genom två SilverStripe moduler, SilverStripe-webservices och ObjectService som är utvecklat av systemutvecklingsföretaget. Webservices modulen implementerar ett REST gränssnitt i SilverStripe. Ett REST gränssnitt erbjuder kommunikation mellan två olika enheter genom användningen av webbteknologi. I figur 12 illustreras kommunikationen mellan frontend och backend med http protokollet. Vanligtvis utvecklas ett REST gränssnitt för kommunikation mellan två olika datamaskiner. Det kan till exempel vara en mobilapplikation eller en bot som behöver hämta eller spara data från databasen genom att kalla på metoder som REST gränssnittet erbjuder. Genom att erbjuda möjligheten för andra enheter att kommunicera med gränssnittet kan olika system med tillgång till samma data utvecklas.

På websystem står att följande designprinciper tillämpas (Websystem u.å):

- ”Varje resurs är unikt adresserbar enligt en gemensam standard (URI).
- Alla resurser har ett gemensamt gränssnitt för att överföra kommandon mellan klient och server. Grundläggande kommandon för att interagera med objekt baserade på de verb som är angivna i HTTP-standarden: POST, GET, PUT och DELETE.
- Varje resurs kan erhållas i ett av flera representationsformat.

- Till skillnad från tjänster implementerade på WS-stacken exponeras data istället för metoder.”

För att implementera CRUD funktionalitet för data som exponeras via REST gränssnittet använde jag mig av ObjectService modulen. Modulen innehåller klasser och hjälpmetoder för CRUD operationer. Genom att ärva ObjectService klassen kunde CRUD funktionalitet tillämpas på data som exponeras.



Figur 12. Illustration över http förfrågningar till REST gränssnittet.

3.3.4 Kommunikation till backend med mobilapplikation

En mobilapplikation utvecklades i samband med webbapplikationen med syftet att kunna ta provexemplar ute på fältet. Mobilapplikationen skulle innehålla funktionalitet för att spara provexemplar och ändra på projektinformation. Kommunikation mellan webbapplikationen och mobilapplikationen var möjligt genom REST gränssnittet. I gränssnittet skapades egna endpoints för mobilapplikationen. Hurdana endpoints som skulle skapas framkom genom diskussion mellan mig och utvecklaren för mobilapplikationen.

Behörigheter hade tillämpats på projekt- och provobjekt. Därav var jag tvungen att implementera autentiseringsmöjlighet för mobilapplikationen. Detta löstes genom användning av Tokens. Token är en sträng data som kan användas för autentisering. Autentiseringsmöjlighet med token fanns tillgänglig i Webservices modulen. En token kunde genereras för en användare genom att kalla en metod som returnerade en krypterad sträng. Krypteringsmetoden lades till som en DataExtension på Member objektet i SilverStripe. På följande sätt kunde en token genereras varje gång en ny användare skapades.

Jag modifierade Controllern för autentisering så att den vid en lyckad inloggning returnerar en giltig token. Med den modifierade Controllern hade mobilapplikationen möjlighet att hämta en token genom en inloggningsförfrågan. Denna token sparas i mobilapplikationen och skickas med i övriga förfrågningar till REST gränssnittet för autentisering.

4. Utvecklingsmiljö

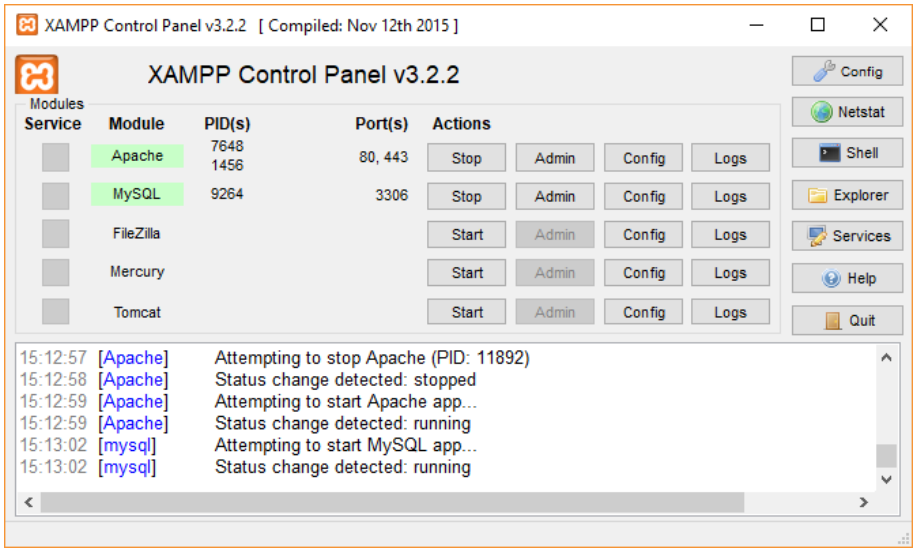
I detta kapitel beskrivs utvecklingsmiljön för projektet. Utvecklingen av webbapplikationen skedde i en lokal miljö. Med en lokal miljö behöver utvecklaren inte lägga fokus på säkerhet eller användning av instabil kod. Utvecklaren kan istället fokusera på att utveckla nya funktioner och kontrollera om de fungerar enligt förväntan. Att utveckla i en lokal miljö har sina fördelar framom extern miljö eftersom utvecklaren har möjlighet att snabbare återställa problem. I en extern miljö är det svårare att återställa problem som uppstår av otestad kod, t.ex. att servern slutar svara. En lokal miljö är säkrare och produktivare.

Webbapplikationens backend hanteras av SilverStripe, vilket gjorde att jag var tvungen att använda en lokal webbserver med stöd för PHP. För att lösa detta använde jag Apache distributionen XAMPP. Distributionen var bekant för mig och jag kunde på kort tid konfigurera servern enligt mina behov. Eftersom jag skulle skriva kod i flera språk valde jag att använda Visual Studio Code som textredigerare. Codes funktionalitet kan utvidgas genom att installera tillägg som är tillgängliga direkt i det grafiska användargränssnittet. Exempel på ett sådant tillägg är XDebug. XDebug tillägget erbjuder ett grafiskt gränssnitt för felsökning i PHP inne i textredigeraren. XDebug behöver även installeras på servern för att Xdebug tillägget skall fungera. Systemutvecklingsföretagets utvecklare använder projektverktyget JIRA, vilket även jag gjorde. Vid utvecklingen av REST gränssnittet använde jag mig av verktyget Postman. För lagring och versionshantering av projektkoden använde jag mig av SourceTree och Bitbucket.

4.1 XAMPP

XAMPP är en Apache distribution som består av fyra grundläggande komponenter, Apache, MariaDB, PHP och PERL. XAMPP fungerar som en webbserver på en lokal datamaskin. XAMPP erbjuder ett grafiskt gränssnitt för att konfigurera inställningar, se figur 13. XAMPP kan köras på Windows, macOS och Linux. Distributioner så som XAMPP tillåter utvecklaren att på kort tid skapa en lokal servermiljö. XAMPP är färdigt konfigurerad för användning efter installation. Genom att använda distributioner som XAMPP förkortas

tiden för installering och konfigurering av enskilda komponenter. I mappen var XAMPP är installerad finns en htdocs mapp tillgänglig. Filer som skall köras på webbservern läggs till i htdocs, t.ex. PHP filer. (Dvorski 2007).



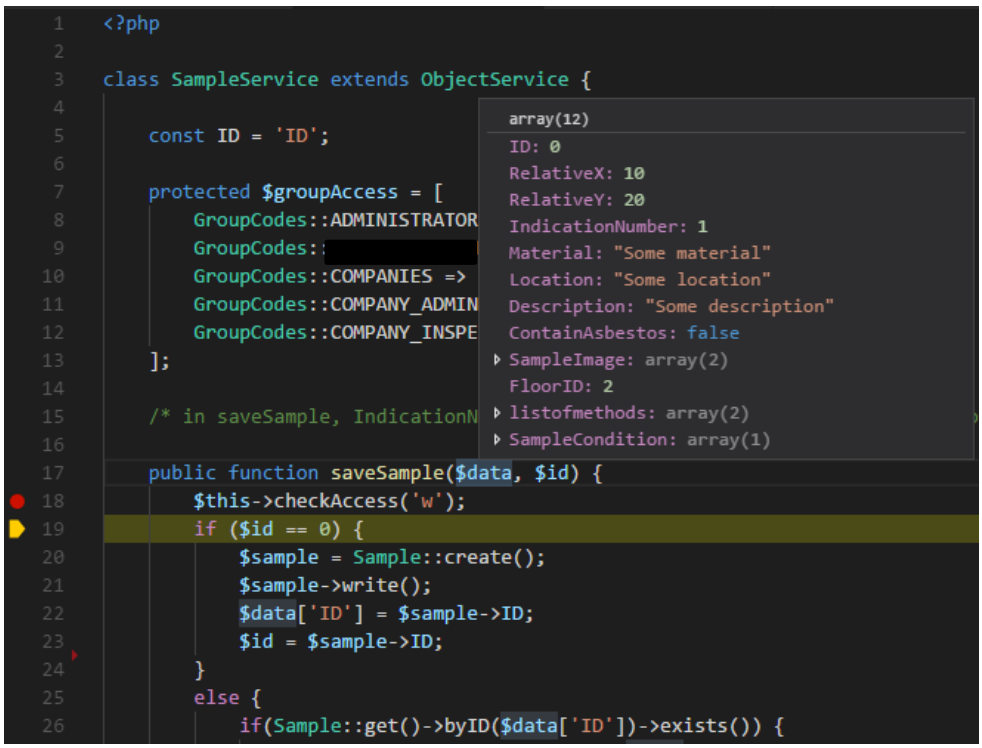
Figur 13. XAMPP kontrollpanel.

4.2 Visual Studio Code

Visual Studio Code är en textredigerare utvecklad av Microsoft. Första versionen av Code släpptes i april 2015. Code är tillgänglig för Windows, macOS och Linux. Code har inbyggt stöd för JavaScript, TypeScript och Node.js. Stöd för andra språk som PHP, Java och C#, kan läggas till genom att installera tillägg. Code erbjuder ett stort bibliotek med många olika typer av tillägg. Det är möjligt att installera tillägg direkt i det grafiska gränssnittet. Code erbjuder kraftfulla editeringsfunktioner så som linting, multi-cursor redigering och IntelliSense. Editeringsfunktioner av denna typ ökar produktiviteten och stöder underhållbar kod. (Francesco 2017).

4.3 XDebug

Xdebug är ett felsökningstillägg som är tillgängligt för PHP. Tillägget erbjuder funktionalitet som underlättar utveckling och felsökning. Ett användbart verktyg är single step debugger. Verktyget tillåter utvecklaren att pausa exekveringen och navigera framåt ett steg i taget. I felsökningsläget har utvecklaren möjlighet att se nuvarande data med manipuleringsmöjligheter. På detta sätt kan utvecklaren följa med koden steg för steg och observera var problem uppstår, se figur 14. Ytterligare funktionalitet som erbjuds är bl.a. stack traces, profiler och remote debugger. (Skvorc 2017).



Figur 14. Felsökning med XDebug i Visual Studio Code.

4.4 Postman

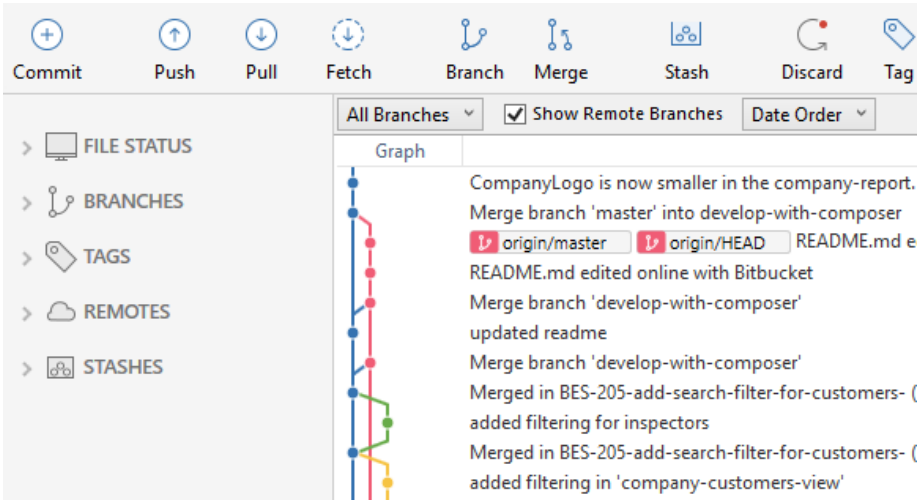
Postman är en HTTP klient som är tillgänglig för Windows, macOS och Linux. Klienten erbjuder ett intuitivt grafiskt gränssnitt med mångsidig funktionalitet. Postman är ett populärt verktyg vid utveckling av API gränssnitt. I gränssnittet kan man konstruera förfrågningar och sedan läsa av responsen i ett tydligt format. Det finns även möjlighet att spara förfrågningar. Postman erbjuder automatisk testning. Automatisk testning innebär att klienten kör flera förfrågningar och sedan kontrollerar om responsen motsvarar eventuella specificerade villkor. Utvecklaren har sedan möjlighet att se vilka förfrågningar som klarade testerna och vilka som inte gjorde det. (Wagner 2014).

4.5 Bitbucket

Bitbucket är en lagringstjänst för Git och Mercurial projekt. Inom systemutveckling är det vanligt med användning av lagringstjänster så som Bitbucket, eftersom det är av stor betydelse att ha koden lagrad på en säker plats. Lagringstjänsten erbjuder möjlighet att se projektkoden i en webbläsare. Tjänsten erbjuder funktionalitet bl.a. begränsa vem som har tillåtelse att se koden och pull requests. Med pull requests kan utvecklaren låta övriga utvecklare inspektera och diskutera förändringar innan de läggs till i projektkoden. (Zivkovic u.å).

4.6 SourceTree

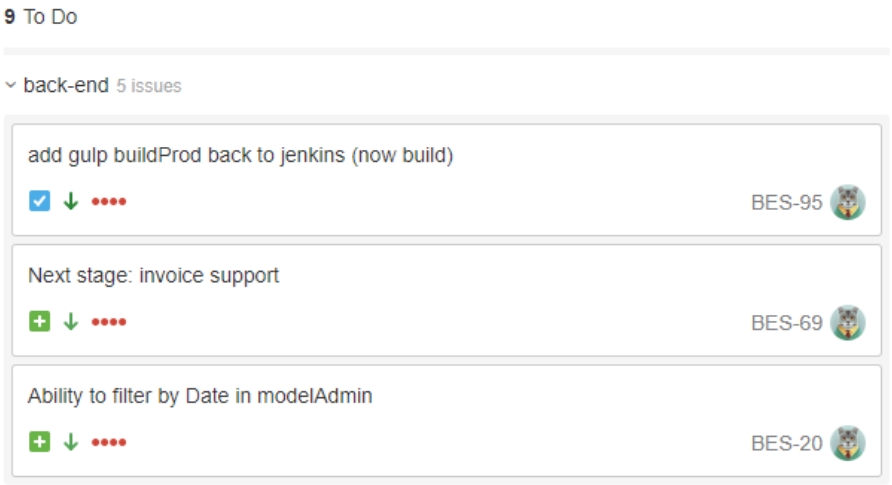
SourceTree är ett grafiskt gränssnitt för versionshanteringssystemet Git. I gränssnittet har utvecklaren tillgång till en översikt av projektkoden. I översikten syns de senaste incheckade ändringarna och grenarna, se figur 15. SourceTree visualiserar processen för Git kommandon. Därmed är det möjligt att steg för steg se vilka ändringar som gjorts i projektkoden. (Kitterhing 2016).



Figur 15. Användargränssnitt i SourceTree.

4.7 JIRA

JIRA är ett projektverktyg för planering och uppföljning av projekt. I JIRA kan samtliga parter se vilka projektuppgifter som är utförda och vilka som skall utföras. Deadlines för uppgifter kan skapas och nya tickets kan tilläggas. I företag där ledningen har kontakt med kunder och utvecklarna endast fokuserar på att utveckla produkten kommer JIRA bra till hands. Ledningen kan lägga till nya tickets beroende på vad som har bestämts vid kundmöten och utvecklaren ser dem i JIRA. Figur 16 visar hur tickets kan se ut i JIRA. Kunden kan också se hur projektet framskrider genom JIRA:s projektöversikt. (Lang 2015).



Figur 16. Tickets i JIRA.

5. Sammanfattning

Projektet har gett bra erfarenhet åt mig som utvecklare. Jag har fått en helhetsbild av hur utveckling av webapplikationer kan gå till. En bra tanke att ha i bakhuvudet när man utvecklar på större projekt är att alltid försöka dela upp funktionaliteten i så små delar som möjligt eftersom det underlättar för vidareutveckling samt förändringar i systemet.

Vidareutveckling av webbapplikationen är under planering. I nuläget har applikationen den funktionalitet som ursprungligen hade avtalats och labbföretagets kunder har tillgång till tjänsten. Detta var min första erfarenhet av SilverStripe och AngularJS. Jag har vidgat mina kunskaper inom webbutveckling som jag tror att jag kommer att ha stor nytta av i framtiden.

I utvecklingsprocessen av webbapplikationen har funnits utmaningar. Vid starten var teknikerna okända för mig. Under processen förekom ständiga förändringar i domänlagret samtidigt som nya funktioner skulle levereras åt kunden. Det resulterade i att jag hade svårt att utveckla en grund som stöder oförutsägbara förändringar. Jag arbetade även med ständig tidspress. Vid förändringar i domänlagret har tiden varit knapp att testa hur projektets funktioner har påverkats.

En särskilt utmanande funktion att implementera i webbapplikationen var automatisk generering av kartläggningsrapporter. Varje rapports innehåll hämtades av ifrågavarande kartläggningsprojekt. JavaScript biblioteket som jag använde för utvecklingen av funktionen som genererar rapporter, innehöll inte funktioner för stöd av dynamiska data. Det innebar att jag var tvungen att implementera stödfunktioner som tar in dynamiska data som sedan

returnerar det i giltigt format för JavaScript biblioteket. Under utvecklingen av rapportfunktionen var det även svårt att göra en felsökning på vad som var den bakomliggande orsaken ifall att en rapport inte kunde genereras.

6. Källförteckning

Achilles, u.å. *Achilles interactive.* [Online]
http://www.achillesinteractive.com/blog/why-do-we-recommend-SilverStripe-cms/ [Hämtat 22 Januari 2018].

Clapp, B., 2015. [Online]
http://brandonclapp.com/what-is-gulp-js-and-why-use-it/ [Hämtat 18 Januari 2018].

Deacon, J., 2005. [Online]
https://6caa0dbc-a-62cb3a1a-s-sites.googlegroups.com/site/urtechfriend/Home/MVC.pdf?attachauth=ANoY7crjzDt8OkDb6kBZ2E8HFhbCduf19AEi9VaG3BXaPV4q_JcsH995sleg6T81SKinU9HuTpH82BzU8DhmPUShiNif-M7DZTOECADQqqfq19T34WXHMCAZU9pTRv8m635SWKQGi4NTNpnit8kTa-VVb7u9TTpI [Hämtat 19 Januari 2018].

Dvorski, D. D., 2007. [Online]
http://dalibor.dvorski.net/downloads/docs/InstallingConfiguringDevelopingWithXAMPP.pdf [Hämtat 16 Januari 2018].

Francesco, H. D., 2017. *Hackernoon.* [Online]
https://hackernoon.com/virtualstudio-code-the-editor-i-didnt-think-i-needed-16970c8356d5 [Hämtat 20 Januari 2018].

Innoweb, u.å. *What is SilverStripe?.* [Online]
https://www.innoweb.com.au/services/SilverStripe-development/what-is-SilverStripe/
[Hämtat 7 Januari 2018].

Kitterhing, J., 2016. [Online]
https://wppusher.com/blog/github-and-sourcetree-throwdown/[Hämtat 15 Januari 2018].

Kukic, A., u.å. *scotch.* [Online]
https://scotch.io/tutorials/angularjs-best-practices-directory-structure
[Hämtat 13 Januari 2018].

Lang, C., 2015. *Clearvision.* [Online]
https://www.clearvision-cm.com/blog/everything-you-need-know-jira-7/ [Hämtat 5 Januari 2018].

Magnusson, S., 2010. *SilverStripe.* [Online]
https://www.silverstripe.org/blog/SilverStripe-cms-the-first-ever-open-source-web-app-to-become-microsoft-certified [Hämtat 24 Januari 2018].

Malik, N., u.å. *MVC and MVVM with AngularJS.* [Online]
https://namitamalik.github.io/MVC-and-MVVM-with-AngularJS/
[Hämtat 13 Januari 2018].

Ringo, Ü., 2015. *TRANSFERWISE TECH BLOG.* [Online]
http://tech.transferwise.com/model-in-angularjs/ [Hämtat 3.1.2018]

Ruebbelke, L., 2015. [Online]
http://onehungrymind.com/reasons-to-love-angular-material/[Hämtat 1 Januari 2018].

Sapegin, A., 2014. [Online]
http://frontendbabel.info/articles/bower-why-frontend-package-manager/[Hämtat 28 Januari 2018].

SilverStripe, u.å. *SilverStripe.* [Online]
https://docs.silverstripe.org/en/3/developer_guides/model/data_model_and_orm/
[Hämtat 10 Januari 2018].

Skvorc, B., 2017. *Sitepoint.* [Online]
https://www.sitepoint.com/getting-know-love-xdebug/[Hämtat 4 Januari 2018].

Uzayr, S. b., 2013. *Noupe magazine.* [Online]
https://www.noupe.com/development/cms/when-to-build-a-site-with-SilverStripe-cms-and-when-not-to-74945.html [Hämtat 4 Januari 2018].

Wagner, J., 2014. [Online]
https://www.programmableweb.com/news/review-postman-client-makes-restful-api-exploration-breeze/brief/2014/01/27[Hämtat 10 Januari 2018].

Websystem, u.å. *Websystem.* [Online]
https://websystem.se/sv/rest-apier
[Hämtat 16 Januari 2018].

Zivkovic, N., u.å. *mogul.* [Online]
https://www.mogul.com/om-mogul/plattformar/atlassian-bitbucket/
[Hämtat 14 Januari 2018].

Bilaga 1 – Enhetssambandsdiagram över projektet, utan entiteternas egenskaper