Prototyp av resurshanteringssystem

- En undersökning av möjligheter för ett företag att hantera inventarier med hjälp av ett resurshanteringssystem som en webbapplikation.

LUNDS UNIVERSITET
Campus Helsingborg

LTH Ingenjörshögskolan vid Campus Helsingborg
Institutionen för datavetenskap

Examensarbete:
Dan Björk

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

Pixelant AB är en webbyrå som utvecklar lösningar för de kunder som behöver webbapplikationer. För att få en överblick över de servrar som hanterar hemsidorna behövde de en prototyp som löste detta problem. Företaget ville veta huruvida de kunde lösa detta problem genom att bygga en webbapplikation med ramverket Angular och därmed följa upp Pixelants inventarier genom API-förfrågningar.

Målet med examensarbetet var att inhämta information kring möjligheten att skapa en webbapplikation för att visa servrar, installationer och tillägg som tillhör Pixelant AB och därmed lösa problemet. Prototypen var en tjänst de behövde, men de var inte säkra på hur de skulle åstadkomma resultatet. Detta examensarbete beskriver metoden som användes och resultatet i form av den prototyp som utvecklades.

Inom de ramar som angivits för examensarbetet kunde en prototyp tas fram där servrar, installationer och installationers tillhörande extensions hos Pixelant AB kunde modifieras, läggas till och tas bort. Med hjälp av det JavaScript-baserade ramverket Angular kunde funktionalitet utvecklas i form av komponenter. Prototypen som togs fram är en single page application som bygger på Typescript, Bootstrap, CSS och HTML. Den back-end som användes under utvecklingsprocessen var en lokal utvecklingsmaskin som tog emot API-förfrågningar och skickade svar. I den skarpa versionen kommer ett API som sköter kommunikation med efterfrågade data kunna användas.

Resultatet innebar att företaget Pixelant, med hjälp av prototypen, kunde modifiera data av inventarier och resurser. Denna prototyp är ett exempel som fler webbyråer kan tänkas ha nytta av med liknande problem.

Nyckelord: Angular 4, JavaScript, Webbutveckling, HTML, CSS, Bootstrap, Resurshanteringssystem

Abstract

Pixelant AB is a web development company developing solution for customers in need of web applications. To keep track of the servers hosting their customer's sites they needed a prototype solving this issue. Pixelant AB wanted to know whether they could solve this issue by building an Angular application to keep track of their inventory such as the server data whilst being able to extend the functionality to keep track of other inventories using API services.

This thesis work's objective was to gather information on the possibilities of creating a web application to present servers, installations and extensions that belongs to Pixelant AB and solving the issue. The web application prototype was a service which they needed, but they were not sure how to achieve that result. The study will present the results on the working strategy and the accomplishment of the prototype.

Within the given framework specified for this thesis work, a prototype was made in which servers, installations and extensions at Pixelant AB could be modified, added and deleted. The functionality of the prototype was implemented using the JavaScript based framework Angular, which was accomplished due to the component feature that Angular has. The prototype that was made is a single page application, which is based on Typescript, Bootstrap, CSS and HTML. The back end for developing purposes that was used was a local virtual machine which was being consumed by the prototype. The production version will be consuming an actual API from the company's resources.

The result of this thesis work meant that the company Pixelant, using the prototype, could modify inventory and resource data. This prototype is an example from which other web agencies with a similar problem may benefit.

Keywords: Angular 4, JavaScript, Web development, HTML, CSS, Bootstrap, Resource management system

Förord

Jag skulle vilja tacka Mattias Nilsson som varit min handledare på Pixelant AB samt de övriga på kontoret för den guidning och hjälp jag fått med arkitekturen av prototypen som byggdes. Det har varit en lärorik och väldigt rolig tid under utbildningen. Jag vill även rikta ett tack till Christian Nyberg och Christin Lindholm som har hjälpt till med rapportskrivning samt att besvara frågor i den inledande fasen av examensarbetet.

Innehållsförteckning

1 Inledning

Detta kapitel introducerar de problem som har behandlats under examensarbetet. Vidare beskrivs varför detta examensarbete valdes och vilka frågor som besvaras.

1.1 Bakgrund

Detta examensarbete har utförts i samarbete med Pixelant AB. Pixelant är en webbyrå och kontoret där examensarbetet utförs ligger i Malmö. Företaget saknar ett system som ger översikt av resurser och vill utvärdera möjligheten att skapa ett sådant system genom att göra en prototyp. Dessa resurser skulle kunna vara Pixelants kunders inventarier, eller Pixelants egna inventarier. Pixelant har flera olika kunder, det är framförallt företag som behöver webblösningar. Inventarierna kan till exempel gälla servrar och information kring dessa.

Prototypen av systemet som byggs ska kunna säkerställa en överblick över dessa inventarier. Prototypen som ska byggas skulle kunna vara en ny typ av lösning för företaget eftersom den använder sig av tekniker som inte prövats tidigare. Prototypen har gjorts med verktyget Angular 2 för att konsumera API-requests från interna API-tjänster. I prototypen ska data om servrar kunna visas, läggas till och ändras. Angular 2 används för att skapa en responsiv single page application [1] och använder sig av komponenter [3]. Prototypen som använder Angular 2 ska vara en webblösning som kan skapa återanvändbara komponenter. Det ska till exempel finnas en komponent för att skapa, editera och visa servrar. Det gäller för fler typer av inventarier på samma sätt som servrar. Data kring inventarierna hämtas genom API-förfrågningar som visas i prototypen. Det finns flera API:er att kommunicera med som tillhandahåller information om till exempel servrar. Svaren från API:erna kan sammanställas med hjälp av verktyg som till exempel GraphQL.

Versionshanteringen Git har använts. Det innebär att alla ändringar som gjorts kommer att utvecklas separat och testas innan de hamnat i prototypen. Flera versioner av prototypen har lagrats i en molntjänst där alla ändringar och versioner sparats.

I framtiden hoppas Pixelant kunna använda och bygga vidare på denna prototyp och använda den för att övervaka inventarierna. För närvarande finns inget system för att göra detta. Det fanns ett tidigare system som inte fungerar längre.

1.2 Syfte och målsättning

Med hjälp av verktygen Angular 2 tas en prototyp av ett system i form av en webbapplikation fram. Syftet med prototypen är att ge en överblick över inventarier på ett enklare sätt än idag. För närvarande finns inget sådant system eftersom det tidigare systemet inte fungerar längre.

Examensarbetets mål är att bygga en prototyp som kan visa data från Pixelants tekniska inventarier och företagets kunders inventarier. Ett sådant system skulle kunna ge företaget möjlighet att utveckla en helhetsbild över inventarierna.

1.3 Problemformulering

Följande frågor besvaras i detta examensarbete:

1. Hur kan en prototyp i form av en webbapplikation göras för att visa inventarier och resurser?
2. Är verktyget Angular 4 som används i prototypen det mest effektiva för ändamålet? Vilka problem ger verktyget upphov till och vilka alternativ finns i sådana fall?
3. Vilka andra verktyg kan vara lämpliga för ändamålet?
4. Hur kan versionshantering användas i denna typ av projekt?

1.4 Motivering inför arbetet på företaget

Datatekniksutbildningen på Lunds Tekniska Högskola ger en bredd i olika tekniker och designprinciper av kodstrukturering. Programmering är ett brett begrepp som innebär mycket och för mig har webbvärlden varit det som verkat mest intressant. Jag var nyfiken på Angular och att utveckla en webbapplikation. Därmed passade detta examensarbete mig bra. Följande kapitel beskriver varför just detta examensarbete valdes.

Jag anser att min utbildning har gett en bredd och gett en bra grund att stå på. Vi har provat på många olika tekniker och programmeringsspråk. För mig har webbutveckling varit det som verkat mest intressant. Jag tror att det är den inriktningen jag vill ha i arbetslivet och tänker att detta examensarbete kan bli en bra första inblick i den branschen.

Eftersom jag är intresserad av att lära mig verktyget Angular och att arbeta i detta verktyg hos ett företag kommer det att bli lärorikt för mig att bli mer förtrogen med den typen av arbete. I framtiden hoppas jag kunna arbeta med webbutveckling och det finns en efterfrågan på denna typ av kompetens. Förhoppningsvis kommer utvecklingen av prototypen ge mig mer kunskap kring ramverket Angular 2. Eftersom flera webbutvecklare på företaget kommer att ingå i projektet hoppas jag även få vidare uppfattning kring för hur det är att jobba i projekt ute i arbetslivet.

För företaget Pixelants del skulle detta examensarbete kunna innebära en början till ett system som kan användas för uppföljning av deras inventarier. Vidare kan de få insikt i huruvida tillvägagångssättet och verktygen för prototypen är adekvata.

Denna lösning skulle kunna vara intressant även för andra som har samma problem med översikt av inventarier. Därmed skulle lösningen kunna gynna mer än bara ett företag.

1.5 Avgränsningar

I detta examensarbete ingår inte arbete mot kund vilket innebär att de tekniska avgränsningarna för prototypen bestäms av företaget.

1.6 Resurser

Det kommer krävas hårdvara för utveckling. Egen dator kommer att användas. Mjukvara kommer att krävas. Utvecklingsmiljöerna är open-source och kommer därmed kunna laddas ned på egen hand. Tillgång till data från API:er kommer tillhandahållas av Pixelant. Det innefattar åtkomst av GitHub-repositories.

En arbetsplats med extern skärm kommer att iordningställas av företaget där utvecklingen kommer att ske. Platsen är i närhet till handledaren på företaget och det finns därmed möjlighet att intervjua och rådfråga handledaren och övrig personal.

2 Teknisk bakgrund

Följande kapitel behandlar den tekniska aspekten av programmeringsspråk som använts i prototypen. Det innebär JavaScript-ramverket Angular och de tillhörande ytterligare verktygen för att framställa prototypen.

2.1 Angular 4

Angular 4, som även numera officiellt kallas för endast Angular [13] [14], är ett modernt JavaScript-ramverk som kan användas för att generera applikations-liknande upplevelser och funktionalitet på webben [1]. Det innebär en hemsida som inte endast innehåller statisk information utan också funktionalitet kan göras med både Angular och JavaScript. Till skillnad mot att använda endast JavaScript ges möjligheten att använda återanvändbara komponenter enligt avsnitt *2.1.1* [3]. Angular bygger på TypeScript som renderas till JavaScript-kod och därmed fungerar i alla webbläsare. [8]

Det går även att manipulera template-koden med hjälp av att injicera Angular-kod varpå template-koden renderas med funktionaliteten. Detta innebär att man exempelvis kan rendera en lista av data, som i annat fall hade krävt mer arbete med vanlig JavaScript-kod. För detta finns alltså en Angular-funktionalitet som används genomgående i prototypen och återanvänds. [7]

Ramverket fungerar för webbläsare och mobila operativsystem såsom Android och IOS. Om resurser från telefonen ska användas bör dock en webbapplikation för telefonen göras om till en Android eller IOS-applikation, vilket kallas att en applikation är native för Android eller IOS och inte körs i mobilens webbläsare utan som en app som laddas ned från Google Play eller App Store.

Det går att göra native mobila applikationer med grunden från ett Angular-projekt med hjälp av ramverken Ionic Framework [10], och NativeScript [11]. Eftersom TypeScipt-kod kan användas i ramverken Ionic och NativeScript går det att återanvända funktionalitet från en tidigare webbapplikation, men även implementera funktionalitet för att använda telefonens resurser [10] [11]. Det innebär att funktioner från telefonen kan användas på samma sätt som en vanlig mobil-applikation. Till exempel kan då kamera, gps och kontaktbok nås, som i vanliga fall anges i manifest-filen i till exempel en Android-applikation [12].

2.1.1 Komponenter i Angular

En viktig del av ramverket är hur applikationen byggs med hjälp av komponenter. En exempelkomponent finns beskriver i *figur 1*. Denna komponent skapar en template som ger en HTML-sida med texten ”Hello World!”. Template kan alltså definieras i TypeScript-filen och därmed i komponenten.

En applikation byggd med Angular består ofta av flera komponenter. Dessa laddas i byggprocessen med kommandot *npm start* med hjälp av en router-komponent som i sin tur endast laddar de komponenter som krävs för den efterfrågade vyn. En komponent har egenskaper för att bygga html-mallar med inbyggd logik som kopplas till dess komponent. Konfigurationen för komponenten kan specificeras. Det innebär att lifecycle hooks [6] för instansieringen kan bestämmas. Vid olika lägen vid instansieringen kan dess egenskaper bestämmas. Vidare kan dessa komponenter kommunicera med varandra, vilket bäst görs med services. [3]

```
1.  @Component({selector: 'greet', template: 'Hello {{name}}!'})
2.  class Greet {
3.    name: string = 'World';
4.  }
```

Figur 1. Exempel på en komponent skriven med TypeScript i Angular. [3]

2.1.2 Services i Angular

En service är en slags komponent som används för att skicka data mellan komponenter och API:er. Det går även att göra genom att skapa komponent-instanser i komponenterna. Problematiken består i att mycket kod kommer upprepas och instansieras i onödan. Service-komponenten kan då innehålla all kommunikation och injiceras i de komponenter som behöver få tillgång till data från andra komponenter. Det här är en funktionalitet som är användbar i många avseenden. Till exempel kan den användas för att hämta de servrar som finns i prototypen. Med hjälp av services, bland annat en server-service i detta fall, kan servrar hämtas från olika komponenter [4]. En exempel-service finns beskriven i *figur 2*. För att så småningom kalla på denna service körs funktionen som visas i *figur 3* i den komponent som vill visa data hämtad från servicen.

src/app/hero.service.ts

```
import { Injectable } from '@angular/core';

import { Hero } from './hero';
import { HEROES } from './mock-heroes';

@Injectable()
export class HeroService {
  getHeroes(): Hero[] {
    return HEROES;
  }
}
```

Figur 2. Ett exempel på en service. [4]

src/app/app.component.ts (getHeroes)

```
getHeroes(): void {
  this.heroes = this.heroService.getHeroes();
}
```

Figur 3. Exempel på hur en service kan användas. [4]

Services används också för autentisering av användare. När en användare ska logga in på applikationen så sparas en token som hamnar i localstorage. Denna kontrolleras vid varje steg då användaren försöker nå någon del av applikationen. Ifall denna token inte stämmer överens med den som finns i localstorage och back-end loggas användaren ut och dess token försvinner. Därefter vidarebefordras användaren till inloggningskomponenten [4].

2.1.3 Routing module

Det finns en instans av en särskild service för att hantera routing. En routing module används i prototypen. Denna fil innehåller de endpoints som pekar på den url där komponenten visas. Detta innebär att olika vyer kan ändras då användaren navigerar i prototypen. Denna endpoint kan användas för att navigera användaren av sidan till de komponenter som efterfrågas. Eftersom alla routing-endpoints finns på samma ställe är de lätta att redigera för utvecklare av Angular-applikationer. Dessa endpoint skapas i en fil *app-routing.module.ts* enligt *figur 4*. Routing-länkar anger komponenter som ska visas vid de specificerade länkarna. I

huvudkomponentens template anges dessa routes med en html-tagg *<router-outlet>* där komponenten visas enligt *figur 5*.

src/app/app-routing.module.ts

```
import { NgModule }             from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

import { DashboardComponent }   from './dashboard.component';
import { HeroesComponent }      from './heroes.component';
import { HeroDetailComponent }  from './hero-detail.component';

const routes: Routes = [
  { path: '', redirectTo: '/dashboard', pathMatch: 'full' },
  { path: 'dashboard',  component: DashboardComponent },
  { path: 'detail/:id', component: HeroDetailComponent },
  { path: 'heroes',     component: HeroesComponent }
];
```

Figur 4. Exempel på en routing modul. [10]

src/app/app.component.ts (template-v2)

```
template: `
   <h1>{{title}}</h1>
   <a routerLink="/heroes">Heroes</a>
   <router-outlet></router-outlet>
 `
```

Figur 5. Exempel på template för huvudkomponent med router-outlet. [10]

2.1.4 Paginering

Det finns en egen komponent som hanterar paginering i prototypen. Komponenten hanterar ett visst antal objekt och numrerar dessa. Komponenten är generisk och kan hantera olika typer av objekt. En färdig design för utseendet av pagineringen kan fås från Bootstrap, men funktionaliteten måste göras med JavaScript eller Angular.

2.2 Bootstrap 4

Komponenternas template-design bygger på Bootstrap version 4. Bootstrap är skapat från början av en tidigare designer från Twitter. Bootstrap används för att

skapa skalbara och färdigdesignade html-element som bygger på CSS. Bootstrap-elementen kan manipuleras i efterhand men innebär en grund att börja på. Det finns färdiga klasser för bland annat formulär, knappar, listor och gruppering av element. Med hjälp av dokumentationen går det att lära sig om de olika element som kan användas för att bygga upp design [15].

2.2.1 Responsivitet

Med hjälp av de element som Bootstrap ger kan prototypen göras responsiv och fungerar därmed oavsett skärmstorlek, hur in-zoomad sidan är och om en telefon eller surfplatta används.

2.3 NPM

NPM används i prototypen för implementering av färdigskrivna Javascript-paket. Till exempel används ett paket för att hämta ikoner till applikationen som heter *font-awesome* [24]. Paketet hämtas med kommandot *npm install font-awesome* i terminalfönstret och hamnar sedan i mappen *node_modules* där alla paketen sparas som senare kommer användas i applikationen. Med hjälp av NPM kan utvecklare samarbeta genom att inte behöva återuppfinna lösningar som någon annan redan gjort. De beroenden som skapas för NPM anges i en fil *package.json* där versionsnummer anges. När kommandot *npm update* körs så uppdateras dessa paket och laddas ned. Det innebär att prototypen får de senaste uppdateringarna som andra utvecklare bidrar med till NPM-paketen [18].

2.4 CSS och SASS

CSS, Cascading Style Sheet, är ett språk som beskriver utseende av HTML och dess element [16]. SASS [17] används för att rendera CSS och för att skapa majoriteten av utseendet i applikationen som inte ärvs från Bootstrap [15]. Det är en annan syntax som innebär att variabler kan användas och när applikationen körs så renderas SASS -filerna. Det innebär att Sass-syntaxen genererar en CSS-fil som applikationen använder. Här kan Bootstrap-elementen överskrivas och manipuleras [16].

2.5 Git och GitHub

Git används för versionskontroll. Git, och GitHub som portal för att nyttja språket, används för att följa upp ändringar som gjorts i prototypen. All tidigare kod som tas bort sparas i form av loggar och likaså det som läggs till. Ett projekt kan startas med hjälp av GitHub och medarbetare kan samarbeta i samma projekt utan att behöva skicka filer emellan varandra när ändringar gjorts. Git följer en typ av trädstruktur som ger olika personer möjlighet att arbeta på individuella uppgifter som inte krockar med varandra. När en feature är färdig kan denna sammanfogas med stammen av projektet, master-grenen. [19]

2.6 Testning och Karma

Utöver manuell testning används egenskrivna tester av Angulars syntax-språk TypeScript med verktyget Karma. Testerna är automatiserade och genomförs automatiskt vid varje pull request samt vid kommandot *npm run test* i kommandotolken. Karma startar en egen webb-server där källkoden körs för de förvalda webbläsarna. Därefter visas vilka tester som genomförts och även huruvida de gick igenom eller inte. [20]

2.7 Wrike

För att skriva user cases och hålla ordning på de tasks som skulle implementeras användes webbapplikationen och verktyget Wrike som Pixelant AB använder till sina projekt. Det finns möjlighet att dela dokument och filer med verktyget samt att skapa uppgifter. Wrike användes för en Scrum-liknande arbetsmetod. Vidare kunde slutanvändare hos Pixelant skriva in sina user cases och därmed beskriva vad de ville att prototypen skulle kunna åstadkomma. [25] [26]

3 Metod och analys

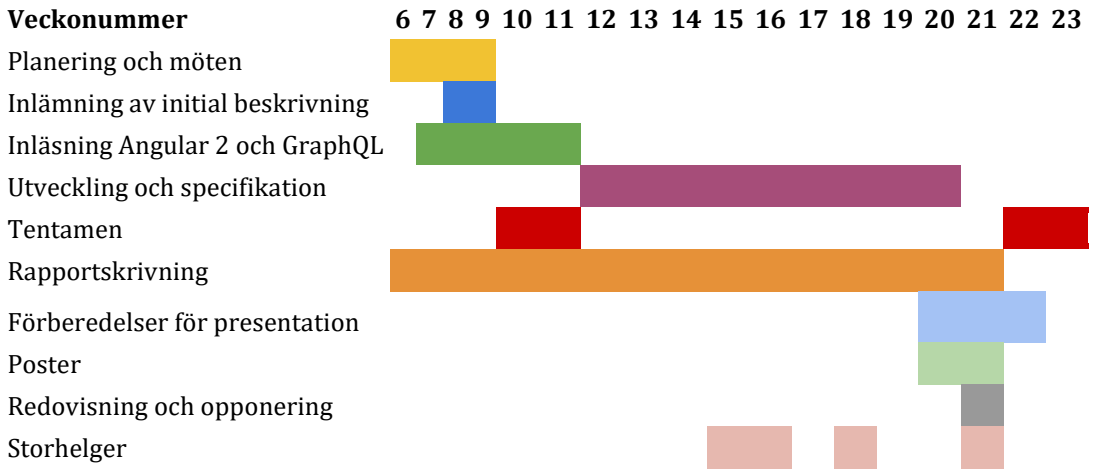
Syftet med följande kapitel är att ge en inblick och analys av den metod som använts under examensarbetets process. Vidare beskrivs de problem som uppkommit och hur dessa har lösts. Hur informationen inhämtats och resultatets analys beskrivs i kapitlet.

3.1 Inlärningsfas

I det första skedet av examensbetet lästes dokumentation in för Angular 2. Eftersom Angular 4-versionen ännu inte hade släppts användes vid denna tidpunkt Angular 2. I dokumentationen finns mycket nödvändig information som används under hela projektets gång. Innan examensarbetet gjordes den tutorial som finns i dokumentationen [9]. Detta var ett förberedande steg som användes för att samla in information kring Angular 2. Tutorialen ger en bild över hur ett projekt sätts upp. Vidare beskriver den funktionalitet som kan implementeras. Detta steg gav en bra utgångspunkt som var uppskattat under veckorna då utvecklingen startade på Pixelants kontor i Malmö.

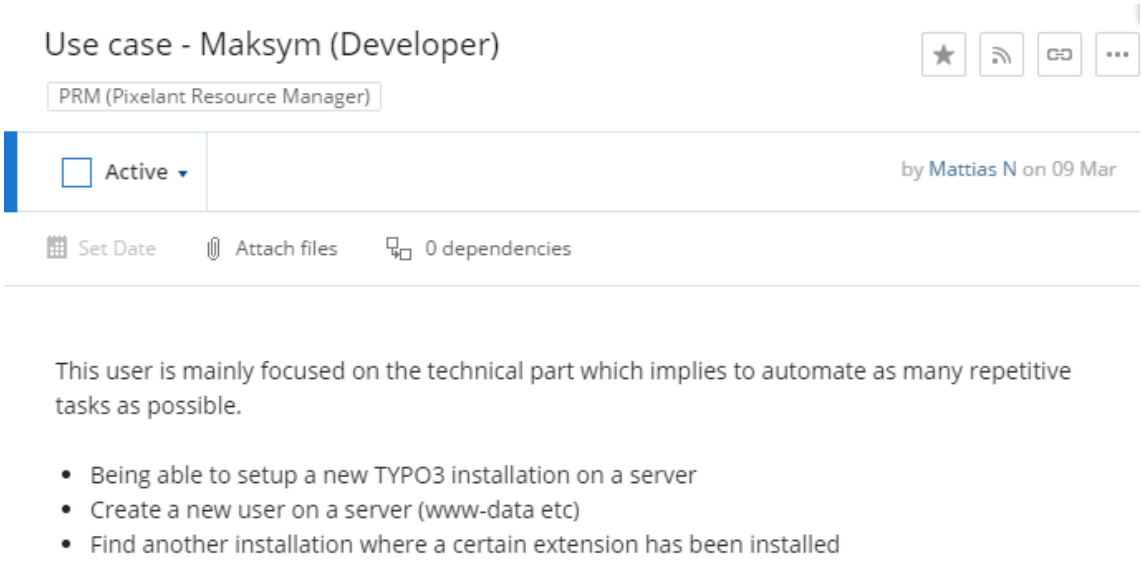
Under den första veckan på Pixelant behövde mycket iordningställas för att kunna starta projektet. Till exempel behövdes SSH-nycklar som kunde användas för att ge tillgång till Pixelants privata repository. En guide användes för att få dessa i ordning. [21]

En tidsestimering gjordes enligt *figur 6*. Det var en övergripande tidsplan över hur examensarbetet planerades att fortgå. Tidsestimeringen gjordes i en inledande fas av examensarbetet för att försöka skapa en bild över hur mycket tid som fanns att disponera på olika delar.

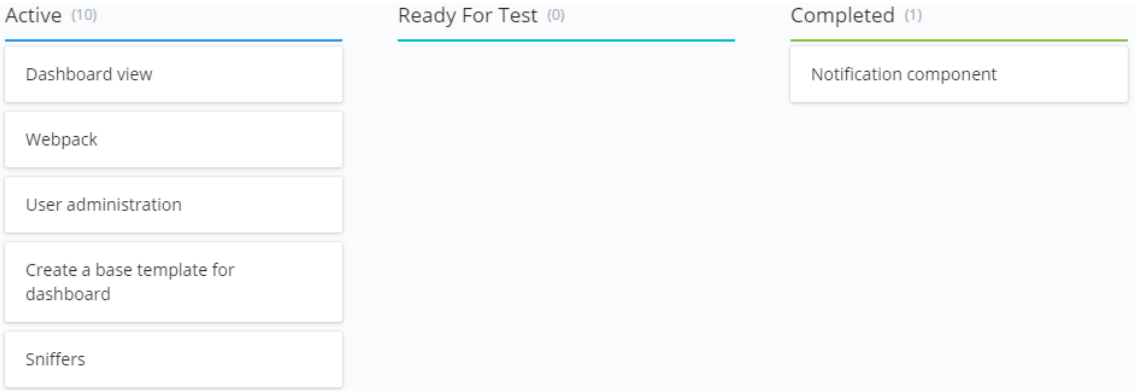


Figur 6. Tidsplan för att ge en överblick i ett tidigt skede av examensarbetet

Mattias Nilsson, projektansvarig och handledare, initierade en projektstruktur och en grund för utvecklingen. Det innebar att projektet för protypen formulerades i Wrike. Där skapades User Cases för de slutanvändare som senare skulle komma att använda prototypen enligt *figur 7*. Tasks (*figur 8*) utformades för att beskriva features som skulle läggas till. Dessa tasks gjordes med inspiration från Scrum. Där bestämdes den funktionalitet som skulle läggas till i prototypen.



Figur 7. Exempel av user case från Wrike, skapat för en av utvecklarna på Pixelant AB.



Figur 8. Exempel av tasks från Wrike. Dessa kunde flyttas beroende på dess status

3.2 Utveckling på Pixelants kontor

Den andra fasen i arbetet innebar inlärning av nya termer och en del tid fordrades till att lära sig den metod som företaget använde sig av för att arbeta i projekt.

3.2.1 Uppstart av projektet
En grund byggdes av Mattias Nilsson för back-end samt front-end. Back-end innehöll det API som Angular-applikationen, prototypen, senare skulle komma att konsumera och kommunicera med. Detta gav en bra bild för att förstå ungefär hur prototypen skulle komma att fungera så småningom.

Med hjälp av GitHub och dokumentationen till prototypens filer kunde utvecklingsmiljön sättas upp. Git användes vidare under projektets gång för att samtidigt kunna arbeta med olika funktionaliteter utan att krocka. När funktionalitet trots allt krockade kunde detta lösas genom att frigöra merge conflicts med hjälp av git.

Varje dag träffades de som arbetade med prototypen och hade ett daily stand up-möte där de involverade beskrev vad de åstadkommit dagen innan mötet och diskuterade vilka eventuella problem de hade. Vidare gicks dagens uppgifter igenom och vilka nya utmaningar som kunde tänkas komma upp.

3.2.2 Utvecklingsprocessen
Angular är ett modernt verktyg och lämpar sig bra för att bygga denna typ av webbapplikationer. Det lämpade sig bra att arbeta med och det fanns oftast lösningar till problemen som uppstod. Med hjälp av dokumentationen som gav exempel på hur vanliga problem kunde lösas gick det att applicera funktionaliteten på prototypen.

3.3 Angular 4 och tillvägagångssätt
Merparten av examensarbetet bestod av att programmera i, samt att lära sig, språket Angular. Eftersom en ny uppdatering släpptes under projektets gång uppgraderades prototypen till den senaste versionen, version fyra.

3.3.1 Komponenter
Med hjälp av funktionaliteten av komponenter i Angular kunde mycket av logiken delas upp inom prototypen. Den första komponenten som skapades var en som innehöll formulär för att skapa en ny server. Det innebar att även en service för att kunna påverka server-data behövde läggas till. Komponenten innehöll template för att beskriva utseendet för formuläret. En funktion ngSubmit() triggas då formulärets data skickas. För att ta reda på hur dessa bör fungera användes Angulars officiella dokumentation [2]. Dokumentationen innehåller också information kring hur validering kan hanteras. [3]

3.3.2 Services
För att kommunicera med API:et och göra delete, update och create-requests användes services. Eftersom dessa endast sköter kommunikationen mellan komponenterna passade det för att separera den logiken i prototypen. [4]

3.3.3 Validering

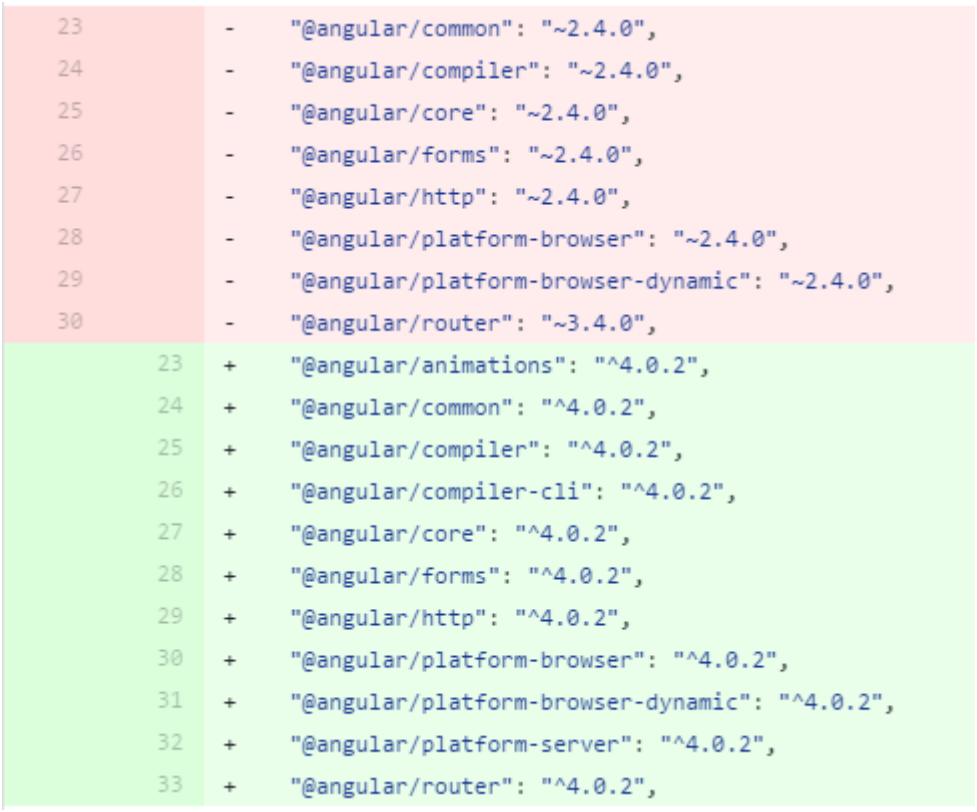
För att garantera att rätt data skickas till API:et lades funktionalitet för validering till. Trots att backend kastar exceptions så är det ur ett användarperspektiv tydligt att redan från början se vad som är av input-data som är fel. Det innebär att ett felmeddelande visas under den input-box som används. I ett första skede gjordes detta med Angulars egen validering för formulär som beskrivs i dokumentationen. Eftersom det visade sig behövas bättre validering för fler typer av input byggdes så småningom en egen komponent för att hantera detta med hjälp av regex [22]. Det innebar att mer komplex validering kunde göras såsom kontroll av giltig url, email och lösenord.

3.3.4 API

Eftersom Docker inte fungerade med git bash på windows och den hostas lokalt valdes att installera en virtuell maskin som körde Linux tillsvidare. Där kördes backend-delen som frontend skickar requests till, varpå Linux-maskinen svarar.

3.3.5 Uppgradering till version fyra

För att prototypen skulle kunna använda de senaste funktionerna uppgraderades Angular-versionen till version fyra som släpptes medan examensarbetet pågick. Det var en process som medförde att mycket av de beroenden som fanns i prototypen fick uppdateras enligt dokumentationen för de nya ändringarna [5]. Dessa ändringar innebar att alla som var delaktiga i git-repositoriet fick installera de nya NPM-paketen som lades till i *package.json* enligt *figur 9*.

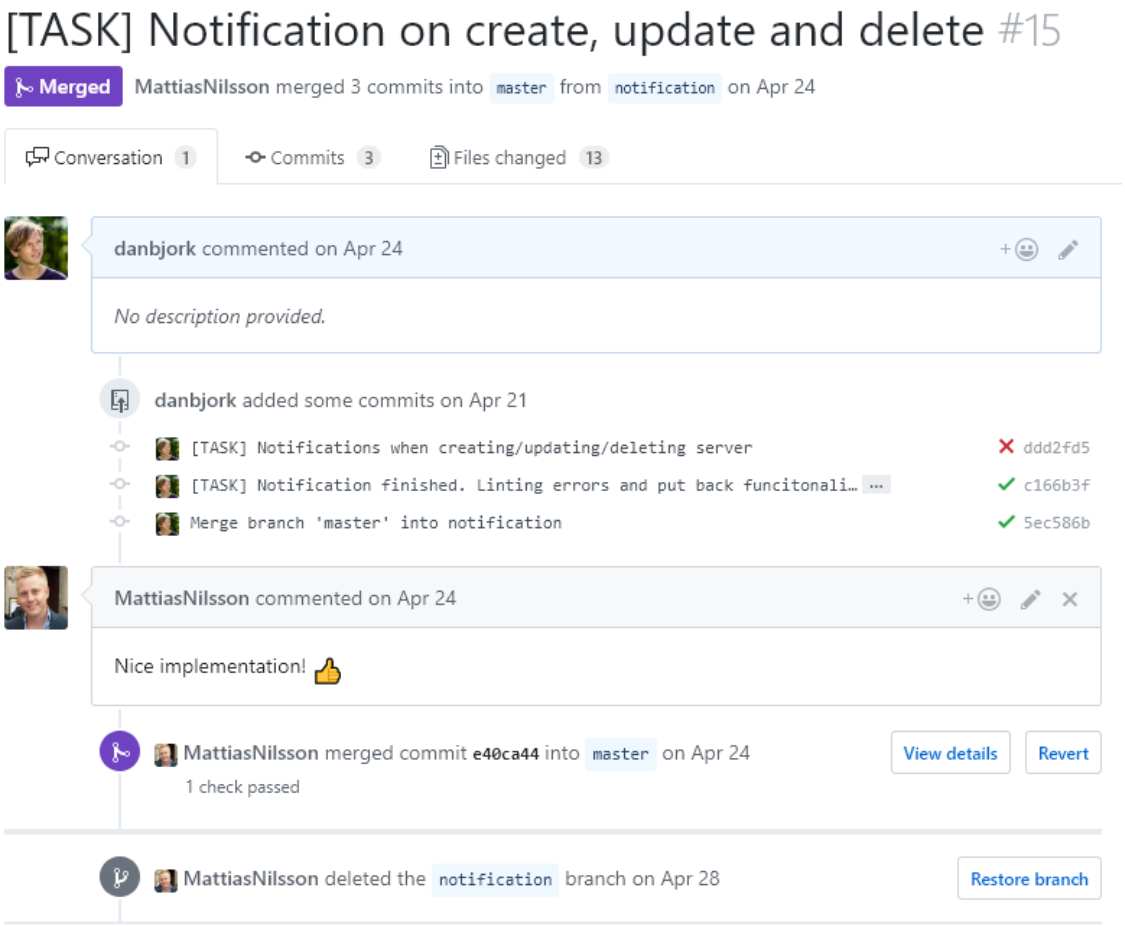


Figur 9. Ändringar som gjordes i package.json efter uppgraderingen. Hämtat från en log från GitHub

3.4 Code reviews och testning

Under hela projektets gång användes GitHub som ger möjligheten att utvecklare granskar kod innan den läggs till master-grenen. Det innebar att alla nya kodrader som lagts till visas, vilket ger komprimerad överblick över de förändringar som gjorts i arbetet. Denna metod kallas *pull request* och innebär en code review innan ändringarna fastslås. Ett exempel på en pull request finns i *figur 10*. Feedback kunde ges på GitHub genom att de rader av kod som granskaren ansåg vara felaktiga kommenterades. Granskningen godkändes inte förrän dessa var återgärdade.

Denna arbetsprocess var densamma som användes hos Pixelant AB. Processen med code reviews var välbeprövad av företaget i tidigare projekt hos dem. Därmed gav det en inblick i hur företag kunde arbeta med granskning av kod internt.



Figur 10. Pull request från implementation av notifikationer

Testningen i examensarbetet skedde i samband med utvecklingen och innebar att felmeddelanden i console-fönstret, efter de ändringar som krävdes gjorts, implementerades för att lösa problemet. Här kunde även data observeras med hjälp av tillägget Augury som beskrev Angular-komponenternas tillstånd och data [27]. Testning gjordes även med hjälp av automatiserade Karma-test. Detta innebar att ett webbläsarfönster startades och kontrollerade att komponenter instansierades och att prototypen fungerade. Karma-testningen kördes också vid varje *commit* i GitHub. Detta visades även vid *pull request* enligt *figur 10* och behövde då åtgärdas innan pull requesten kunde godkännas. Vidare användes linting som innebar att kodsyntax strukturerades på samma sätt i hela prototypen. Det innebar att inga extra tomma rader fanns i koden och att indenteringen i koden var generisk.

3.5 Slutfasen av examensarbetet

I slutfasen av examensarbetet sammanställdes den inhämtade informationen och rapportdelen slutfördes. Vidare skrevs delar såsom resultat och analys. En poster skapades och presentation av arbetet färdigställdes.

3.6 Källkritik

I följande avsnitt diskuteras de källor som använts i examensarbetet.

3.6.1 Källor hämtade från dokumentation

Dessa källor är de beskrivna i avsnitt *7.1* (källorna [1] – [9]) samt källorna [10] – [12] och [15] – [20] samt [24] – [29].

Dessa källor är de som angivits från direkta länkar till ett verktygs dokumentation bör vara de bästa möjliga för ändamålet eftersom denna är skriven av de som utvecklat verktyget eller hjälpmedlet. Till exempel innehåller dokumentationen för Angular en fullständig applikation med fungerande exempel som innehåller mycket av det som en webbsida kan behöva. Eftersom dokumentationen är skriven av utvecklarna eller i samarbete med utvecklarna är denna typ av källa sannolikt den mest trovärdiga. Innehållet är väl beskrivet och används ofta som referens på forum såsom StackOverflow för att beskriva lösningar på olika problem.

3.6.2 Övriga källor

De övriga källor som använts för begrepp och nyheter kring ny funktionalitet som har använts är inte säkra källor på samma sätt som de som är från dokumentation.

[13] [14] Dessa källor användes som komplement för att informera kring de nya uppdateringarna till version 4 av Angular. Dessa ändringar kunde bekräftas till exempel på den officiella Angular-sidan med hjälp av en change log de gett ut [5]. De anses därmed sannolikt vara trovärdiga. [14] Är dessutom en blogg skriven av Angular.

[21] Denna källa angavs av handledaren Mattias Nilsson för att starta upp projektet som vidare gav åtkomst till de resurser som krävdes. Webbplatsen används för att förenkla infrastrukturen för utvecklare. Källan användes dock inte för utvärdering av problemställning utan som en beskrivande del i metod- och analyskapitlet.

[22], [23] Användes för att ytterligare bekräfta påståenden kring hur API och REGEX fungerar. [22] Är hämtad från Microsoft och anses därmed vara trovärdig. [23] Denna källa är avsedd att hjälpa utvecklare genom utbildning på internet. Avsikten för MakeUseOf är att informera kring teknikvärlden. [23] Den här källan

anses inte vara trovärdig i samma utsträckning som de som är tagna direkt från ett verktygs dokumentation.

3.7 Analys

I följande avsnitt diskuteras besluten som gjorts kring de valda ramverken, problem och dess lösningar. Vidare analyseras den arbetsmetod och de verktyg som använts.

3.7.1 Analys av de verktyg som använts för utveckling

I detta avsnitt analyseras de olika verktyg som använts i prototypen. Vidare diskuteras alternativ samt deras fördelar och nackdelar.

3.7.1.1 Analys av Angular som val av JavaScript-ramverk

För att göra samma typ av applikation i ren JavaScript hade dels mer arbete krävts, dels hade de funktioner som Angular ger saknats. Till exempel hade det varit svårt att strukturera prototypens kod i form av komponenter, som redan finns i Angular. Det hade i sin tur lett till svårigheter med att lägga till ny funktionalitet. Eftersom komponenter är självständigt fungerande och inte nödvändigtvis beroende av varandra går det med hjälp av Angular att lösa problemet, vilket ger en skalbar kod. Angular gav därmed möjlighet att låta prototypen växa, även när den var färdigställd.

En utmaning med Angular är det faktum att projektet kan växa sig mycket stort och komplext om slutprodukten innehåller enkel funktionalitet. Huruvida det är värt att använda sig av Angular beror på syftet och på vilken typ av komplexitet slutprodukten kräver. För att kunna genomföra detta examensarbete krävdes en komplexitet av slutprodukten som rättfärdigade valet av Angular. Komplexiteten innebar att det fanns en back-end som skulle hanteras och att funktionalitet återanvändes i form av komponenter i prototypen. Det hade därmed varit svårare att skapa en prototyp i denna storlek med endast JavaScript.

Eftersom det finns många JavaScript-ramverk går det inte att fastslå att Angular var det ramverk som på mest effektivt sätt löste de problem som Pixelant angivit. Till exempel finns ramverken Vue.js och React.js som tillhandahåller funktionalitet som komponenter [28] [29], vilket var en vital del för utvecklingens effektivitet i skapandet av prototypen. Något av dessa ramverk skulle ha kunnat vara alternativ till Angular.

Det är viktigt att välja ett ramverk och hålla sig till det eftersom det finns en uppsjö av JavaScript-ramverk. En förstudie hade kunnat göras för att hitta ett bättre alternativ till Angular. Efter inläsning av dess dokumentation i uppstarten av examensarbetet visade det sig emellertid att verktyget var kraftfullt nog att klara den funktionalitet som efterfrågades för prototypen. Angular som JavaScript-

ramverk bestämdes av handledaren Mattias Nilsson. Detta ramverk valdes eftersom det använts av handledaren i tidigare projekt.

Det fanns nackdelar med ramverket som visade sig under utvecklingsfasen av examensarbetet. Till exempel behövdes nya komponenter läggas till app-module-filen för att kunna användas. Det innebar extra arbete oavsett komplexiteten av den komponent som skulle läggas till. Fördelen med instansieringen var den förståelse för hur projektet var strukturerat genom att genomföra detta momentet. En ytterligare nackdel med Angular var den refaktorering som krävdes vid uppgraderingen till version 4. Det innebar att när komponenterna som var beroende av varandra flyttades enligt den nya projektstrukturen behövde de omdefinieras. Detta är ett problem som är komplicerat och som återkommer i andra JavaScript-ramverk.

3.7.1.2 Analys av Bootstrap

För att inte behöva återskapa CSS-design för html-taggar användes Bootstrap. Bootstrap användes som grund på vilken de egenutvecklade elementen byggde på. Det innebar att färdig funktionalitet kunde återanvändas, vilket var en fördel för utvecklingen av prototypen. För att skapa eget utseende kan dessa taggar överskridas, vilket innebär att ett elements skalbarhet kan behållas medan egenskaper som färg och utseende kan ändras. Överskrivande av Bootstrap-element gjordes för att skapa den design som Pixelant AB ville ha. För att kunna göra alla olika element från grunden hade det krävts ytterligare tid och kunskap av CSS. Att använda Bootstrap var alltså ett val som gjordes ur effektivitetssynpunkt och samtidigt gav det önskade resultatet.

3.7.2 Analys av arbetsprocessen

För att kompensera för bristfällig erfarenhet av utveckling i Angular och i webbutveckling krävdes grundlig och därmed tidskrävande inläsning i uppstarten av examensarbetet. Dessbättre gav dokumentationen till Angular en god insikt i hur ramverket fungerade. I detta tidiga skede hade det varit fördelaktigt att få en bredare insikt i hur Angulars språk TypeScript skiljer sig från JavaScript istället för att lita på tidigare erfarenhet av JavaScript.

Arbetsprocessen skedde enligt det arbetssätt som användes hos Pixelant. Det innebar en metod som inte är definierad, men kan beskrivas som agil utveckling. De viktigaste tasks eller problem med prototypen angavs vid morgonens stand-up möte som genomfördes varje dag under utvecklingsfasen. Det innebar ett flexibelt arbetssätt för att genomföra det som var viktigast. Prototypens funktionalitet kunde till exempel ha specificerats enligt vattenfallsmetoden, men eftersom prototypens krav ofta ändrades så hade vattenfallsmetoden förmodligen fungerat sämre än den valda agila metoden. Det fanns också önskade tasks som tyvärr inte hann implementeras inom ramen för examensarbetet. [30]

De code reviews som genomfördes genom att Mattias Nilsson och andra kollegor kommenterade *pull requests* var en metod som Pixelant AB använt i tidigare projekt. För att ytterligare testa prototypen hade tester kunnat skrivits på förhand. Det hade kunnat vara ett dokument som till exempel specificerade hur validering skulle skötas. I ett senare skede hade då en testare kunnat följa dokumentet i kombination med prototypen. På det viset hade fler problem med prototypen kunnat hittas istället för att testning av en implementerad funktion testades under utvecklingen.

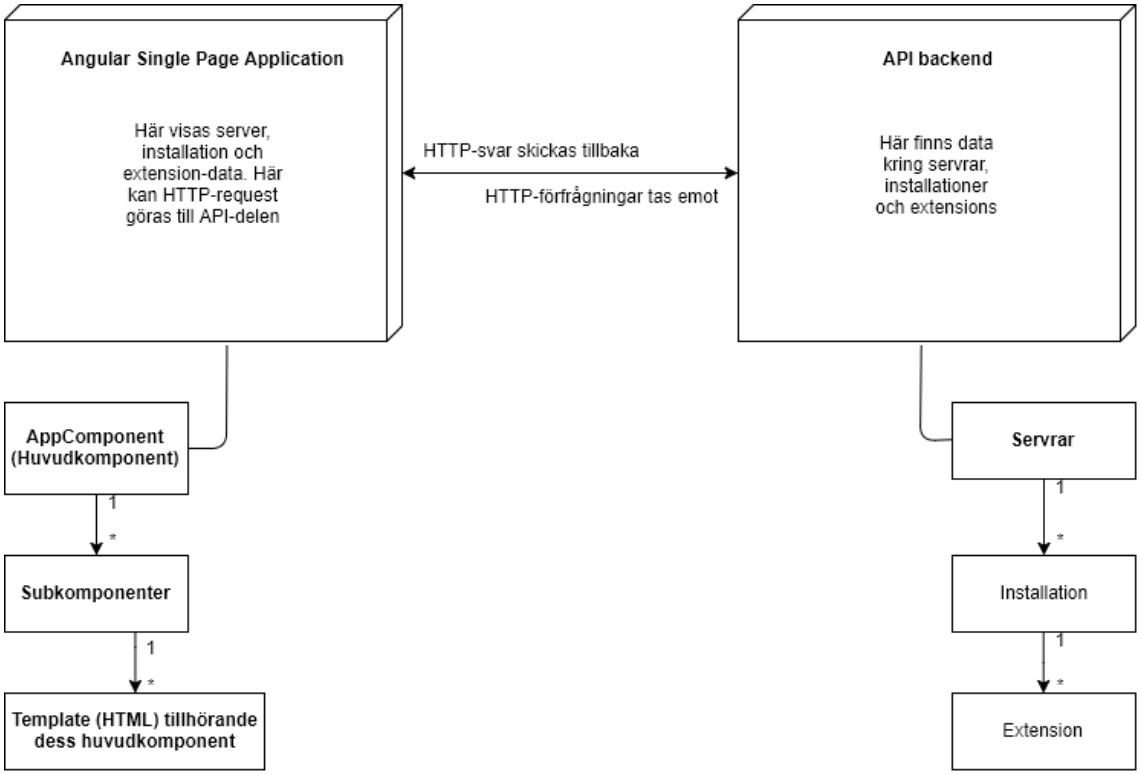
De tester som var automatiserade med hjälp av verktyget Karma hade också kunnat utvecklas genom att skriva mer specifika tester, enligt dokumentationen för Karma. Exempel på tester finns även beskrivna i Angulars dokumentation. På grund av den begränsade tidsramen prioriterades detta bort.

4 Resultat

Detta kapitel innehåller resultat från utvecklingen av prototypen. Den färdiga prototypen beskrivs i detta avsnitt.

4.1 Översikt av prototypen

Den färdiga prototypen är utformad som ett träd av komponenter och en huvudkomponent, *AppComponent*. Vidare finns flera *services* precis som det finns fler komponenter. Dessa sköter kommunikationen med API-delen. Prototypen är utformad enligt *figur 11* som beskriver hur de olika delarna i prototypen hänger samman. I API-delen finns data och http-request kan göras till denna del från front-end. I front-end visas, skapas och tas data bort genom att skicka förfrågningar via http till back-end.



Figur 11. Överblick över projektets struktur

4.2 Prototypens funktionalitet

Följande avsnitt beskriver utseende (med tillhörande figurer i appendix) av prototypen samt dess navigering.

4.2.1 Utloggad användare

Ett inloggningsformulär (*figur 12*) visas oavsett vilken komponent eller url som försöker nås ifall användaren inte är inloggad eftersom detta kräver en giltig token som sparas i local storage. När användaren loggat in dirigerar prototypen användaren till prototypens dashboard (*figur 4A i appendix*).

Figur 12. Inloggningsformulär som visas innan användaren är inloggad

4.2.2 Inloggad användare

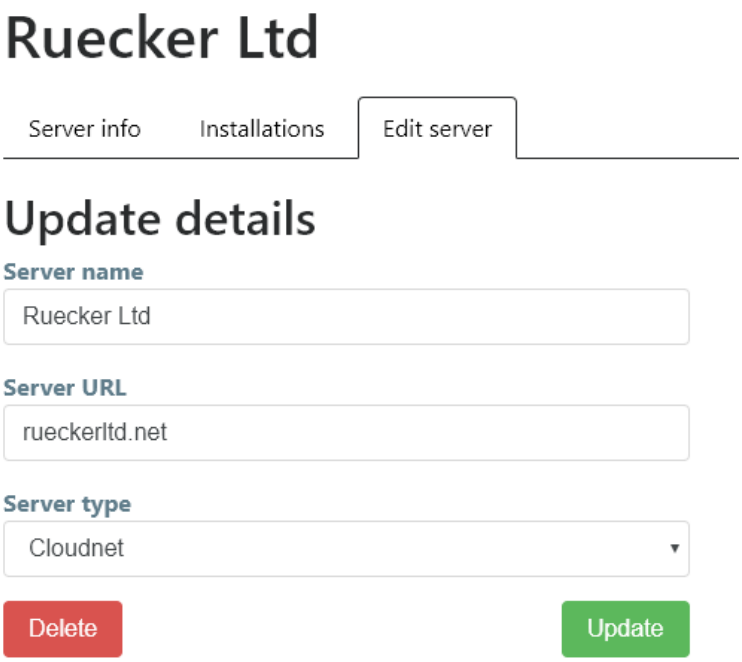
Som inloggad användare visas prototypens dashboard. Den innehåller en översikt av data i form av statistik som är lagrad på back-end-delen. Det innefattar antal servrar som lagts till, de unika installationer som finns, antal extensions tillhörande en installation och antal användare. Denna vy kan ses i *figur 4A*.

En sidomeny finns för att ge åtkomst till de servrar som finns *figur 1A (Appendix)*. I sidomenyn visas och nås även servrars tillhörande installationer genom en toggle-pil. För att kunna navigera i prototypen används element som triggas i prototypens routing-module. När en användare till exempel klickar på en server så visas en servervy-komponent innehållande serverns data. En användare kan klicka på en server och navigeras därefter till servern *figur 2A*.

Det finns ytterligare en väg att nå servrar och installationer via topp-menyn som visas i *figur 19*. Denna meny visar en listvy av den data som efterfrågas. Om en användare till exempel klickar på ”Servers” visas en listvy av dessa som pagineras enligt *figur 3A*. I denna listvy kan specifika servrar visas på samma sätt som de visas via sidomenyn enligt *figur 3A*. En användare kan också se tillhörande

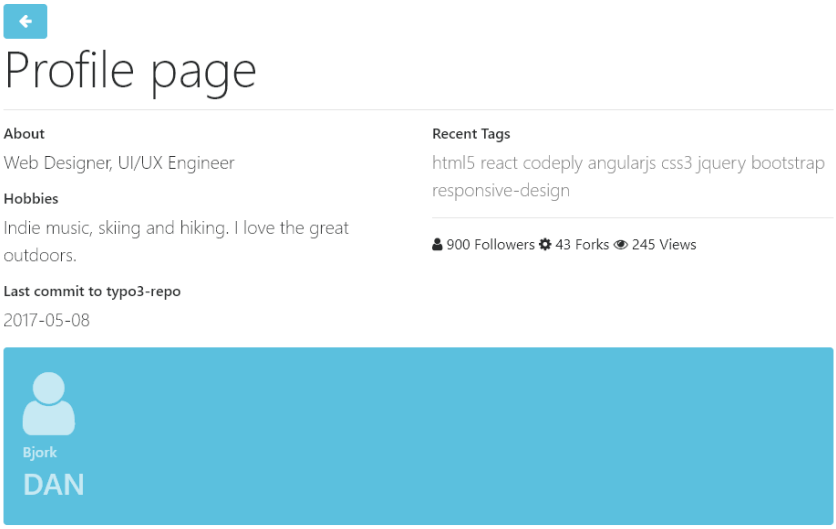
installationer och extensions i denna vy. Dessutom kan användaren uppdatera server-information enlig *figur 14*.

Figur 13. Topp-meny som ger en list-vy med paginering



Figur 14. Formulär för att manipulera server-data

Via toppmenyn kan en användare också se andra registrerade användare i en listvy på samma sätt som i vyn för servrar och installationer. Genom att klicka på användarens användarnamn (*figur 13*), i detta fall ”dev”, visas användarens profil-sida (*figur 15*). Genom att klicka på ”dev” finns också möjligheten för användaren att logga ut i drop down-listan. Då tas användarens session bort i localstorage och användaren dirigeras till inloggningskomponenten (*figur 12*). Ingen av de andra komponenterna tillåts visas med hjälp av en *LoggedInGuard* vars lösning beskrivs i avsnitt 4.3.3.



Figur 15. Vy som visar den inloggade användarens profil

4.3 Specifika komponenter

Följande avsnitt beskriver komponenters funktion och hur implementeringen av dessa lösts i prototypens kod.

4.3.1 Formulär

För att skapa en ny server används ett formulär enligt *figur 16* i template-filen. Ett formulär beskrivs med hjälp av en tagg *<form>*. Taggen för indata, *<input>*, beskriver objektets data. Enligt *figur 17* läggs serverns titel till. Denna titel kan emellertid inte skickad förrän all efterfrågad data lagts till och valideringen är godkänd och submit-knappen går att klicka på. När en användare klickar på submit-knappen körs metoden *ngSubmit()*. Detta kan endast göras då formulärets validering är godkänd, detta beskrivs i template-filen enligt *figur 17*. Eftersom template-filen kommunicerar med dess typescript-fil kan metoderna skrivas till HTML-koden. Till exempel körs en metod *goBack()* som skickar användaren tillbaka till den tidigare sidan.

```
<form [formGroup]="serverForm" (ngSubmit)="onSubmit()">
    <div class="form-group">
        <label for="title">Title</label>
        <input class="form-control" formControlName="title"
        placeholder="Title" [(ngModel)]="server.title" id="title" />
        <control-messages [control]="serverForm.controls.title">
        </control-messages>
    </div>
```

Figur 16. Template för att skicka formulärdata

```
    <button type="submit" class="btn btn-success pull-right"
     (click)="createServer()" [disabled]="!serverForm.valid"
    >Submit</button>
    <button class="btn btn-secondary pull-left" (
    click)="goBack()">Back</button>
</form>
```

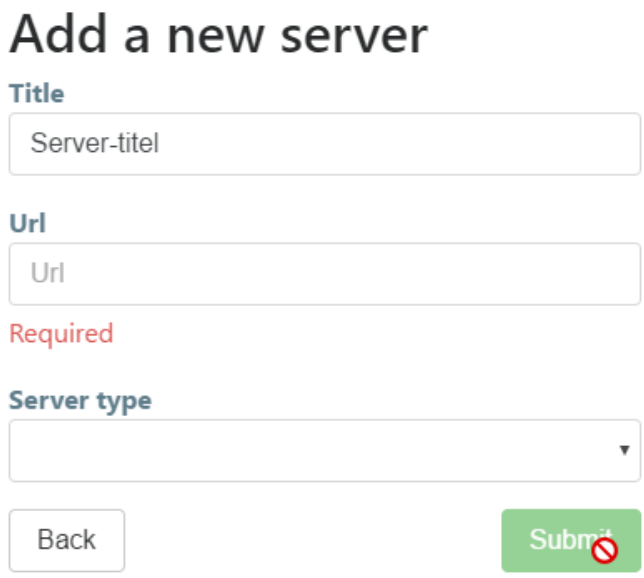
Figur 17. Beskrivning för hur formuläret skickas i template-filen

För att skapa ett formulär i typescript-filen måste modellens fält läggas till i *<input>* och därmed *[(ngModel)]*. När fälten är godkända kan metoden *createServer()* köras i typescript-filen. Då skickas server-objektet så småningom till *ServerService* som gör ett http-request för att lägga till servern enligt *figur 18*.

```
create(server: Server): Promise<Server> {
    return this.http
        .post(this.serversUrl, server,
            this.httpClient.authOptions())
        .toPromise()
        .then(res => res.json())
        .catch(this.handleError);
}
```

Figur 18. Create-metoden i filen server.service.ts

4.3.2 Validering av formulär



Figur 19. Exempel på ett formulär i prototypen som inte kan skickas såvida fälten inte är korrekt ifyllda

Valideringens funktion beskrivs visuellt genom en skärmdump i *figur 16.* Formuläret kan endast skickas då en titel för en server är ifylld, annars går det inte att klicka på submitknappen. Ett felmeddelande visas om användaren klickar i input-fältet men inte skriver någon titel. Detta gäller för alla input-fält som kräver att data skickas för hela prototypen. Vidare visas en särskild text om skrift lagts till i till exempel url-fältet med en indata som inte är en giltig url. Detta löstes med regex-kod som måste matchas med indata. Ifall denna indata inte stämmer överens med det efterfrågade värdet returnerar applikationen att url som angivits inte godkänns. Lösningen visas i kodexemplet i *figur 17*.

```
static urlValidator(control: FormControl) {
    if (control.value) {
        if (control.value.match(/[-a-zA-Z0-9@:%_\+
            return null;
        } else {
            return { 'invalidUrlAddress': true };
        }
    }
}
```

Figur 20. Regex-kontroll och kod som returnerar huruvida en url är giltig eller inte

4.3.3 LoggedInGuard

LoggedInGuard är en komponent vars funktion används för alla de övriga komponenterna. För att säkerställa att inte åtkomst ska ges till prototypen utan att en användare är inloggad, används LoggedInGuard för att blockera och aktivera komponenter. Denna lösning innebar att en url som till exempel ”/dashboard” i prototypen dirigerade användaren till inloggningsformuläret om användaren inte var inloggad. Komponentens lösning i form av TypeScript-kod beskrivs i *figur 21*.

```
import { LoginService } from './shared/login.service';

@Injectable()
export class LoggedInGuard implements CanActivate {

    constructor(
        private loginService: LoginService,
        private router: Router) {}

    canActivate() {
        if (!this.loginService.isLoggedIn()) {
            this.router.navigate(['']);
        }
        return this.loginService.isLoggedIn();
    }
}
```

Figur 21. LoggedInGuard-komponenten

Denna lösning implementerades vidare i routing-modulen genom att *canActivate()-metoden* lades till den routing som gav en komponent. Lösningen i routing-modulen beskrivs enligt *figur 22*.

```
const routes: Routes = [
    { path: 'dashboard', component: DashboardComponent, canActivate: [LoggedInGuard] },
    { path: 'servers', component: ServerComponent, canActivate: [LoggedInGuard] },
    { path: 'servers/:id', component: ServerDetailComponent, canActivate: [LoggedInGuard] },
    { path: 'addserver', component: ServerFormComponent, canActivate: [LoggedInGuard] },
    { path: 'installations', component: InstallationsComponent, canActivate: [LoggedInGuard] },
    { path: 'installations/:id', component: InstallationDetailComponent, canActivate: [LoggedInGuard] },
    { path: 'addinstallation', component: InstallationNewComponent, canActivate: [LoggedInGuard] },
    { path: 'addinstallation/:serverId', component: InstallationNewComponent, canActivate: [LoggedInGuard] },
    { path: 'sidebar', component: SidebarComponent, canActivate: [LoggedInGuard] },
    { path: 'profile', component: ProfileComponent, canActivate: [LoggedInGuard] },
    { path: 'users', component: UsersComponent, canActivate: [LoggedInGuard] },
    { path: 'user', component: UserNewComponent, canActivate: [LoggedInGuard] },
    { path: 'user/:id', component: UserEditComponent, canActivate: [LoggedInGuard] },
    { path: '**', redirectTo: '' }
];
```

Figur 22. Routes i filen app-routing.module.ts

.

5 Slutsats

Följande avsnitt innehåller en sammanfattning av det viktigaste i resultatet samt svar på de frågor som formulerades i problemformuleringen. Kapitlet innehåller beskrivning kring hur prototypen bidrar till att uppfylla syftet med examensarbetet och hur prototypen kommer att användas i framtiden hos Pixelant AB.

Enligt de undersökningar och den utveckling som gjordes kunde slutsatsen dras att Angular fungerade som verktyg för att skapa prototypen och därmed uppfylla de mål som fanns definierade i den inledande fasen. Syftet att ge en överblick över inventarierna (servrar, installationer och extensions) kunde åstadkommas med hjälp av prototypen. Det gick därmed enklare att få en överblick över inventarierna efter att prototypen färdigställts än tidigare. Därigenom kunde Pixelant få en tydligare överblick och helhetsbild över inventarierna. Följande frågor från problemformuleringen besvaras i ordning i avsnitten *5.1*, *5.2*, *5.3* och *5.4*.

1. Hur kan en prototyp i form av en webbapplikation göras för att visa inventarier och resurser?
2. Är verktygen Angular 4 som används i prototypen det mest effektiva för ändamålet? Vilka problem ger detta upphov till och vilka alternativ finns i sådana fall?
3. Vilka andra verktyg kan vara lämpliga för ändamålet?
4. Hur kan versionshantering användas i denna typ av projekt?

5.1 Webbapplikation skapad för inventarier och resurser

För att åtgärda problemet med att visa inventarier och resurser kunde ramverket Angular användas. Utöver att endast visa de inventarier och resurser företaget Pixelant AB hade kunde även data manipuleras och ändras på ett och samma ställe. Med hjälp av den dokumentation som Angular har beskriven kunde de rätta metoderna användas för att åtgärda de hinder som uppkommit.

Prototypen möjliggjorde kommunikation med de API-förfrågningar som så småningom kom att skickas från de olika källor som Pixelant efterfrågade att kunna inhämta data ifrån. I slutskedet skedde kommunikationen fortfarande med det egenbyggda back-end som skulle ersättas av det riktiga API:et. Eftersom data inte bör manipuleras under utvecklingsstadiet valdes detta som en lösning tills vidare. I slutfasen av examensarbetet började Pixelants koppling till Wrikes API att implementeras för att så småningom kunna användas.

Det fanns möjlighet att utveckla denna prototyp på andra plattformar. Till exempel hade prototypen kunnat utformas som en applikation som inte nödvändigtvis var gjord endast för webben. Problemet hade då krävt mer kompetens för att underhålla denna då teamet på Pixelant har mest erfarenhet inom webbutveckling. Istället fanns nu möjlighet att använda exempelvis Ionic för att bygga om ett tidigare projekt i Angular och därigenom skapa en native mobil-applikation. Vidare

fungerade applikationen i telefonen eftersom Bootstrap-elementen anpassar sig och är skalbara utefter den upplösning som användaren använder sig av.

Prototypen är alltså ett exempel på hur ett företag kan hantera inventarier och hur resurser kan lösas. Prototypen som skapades i form av en webbapplikation kunde genomföras med hjälp av ramverket Angular.

5.2 Effektivitet, problem och alternativa lösningar till problem

Ett problem med prototypen var att få asynkrona life cycle hooks att laddas i rätt ordning. Det innebar att data från hämtad från API:et inte nödvändigtvis fanns tillgänglig då en komponent instansieras. Template laddas ibland före *ngOnit()*-funktionen och data hinner därmed laddas i TypeScript-filen. Detta är ett känt problem som verkade vara svårt att lösa i den version av Angular som användes. Till exempel visas inte dashboardens grafer direkt efter inloggning utan först när routing sker. Detta är ett Angular-problem som är svårt att lösa då denna komponent är beroende av externa npm-paket. Vissa av dessa problem genererar med andra ord svåra lösningar som möjligtvis hade kunnat lösas bättre med till exempel ett annat JavaScript-ramverk såsom Vue.js eller React.js.

Lösningen till problemet att data inte hann laddas innan instansieringen, löstes genom att spara API-data till *AppComponent*. Detta var en lösning som med mer Angular-erfarenhet möjligtvis hade kunnat lösas på ett bättre sätt. Eftersom ett nytt API-request körs i *AppComponent* trots att data inte har ändrats skapades ett obefogat moment ur en effektivitetsaspekt. Möjligen är detta ett problem vars lösning fungerar mer effektivt med hjälp av ett annat JavaScript-ramverk än Angular. Det finns flera JavaScript-ramverk som förmodligen skulle kunna ge en mer effektiv implementation. Till exempel skulle alternativen Vue.js och React.js kunnat användas för utveckling av prototypen. Angular är alltså inte det ramverk som är mest effektiv utan kan ersättas av till exempel Vue.js eller React.js. Detta är dock till ett pris som är värt att betala för ett mer komplext prototyp. [28] [29]

5.3 Lämpliga verktyg

Vue.js och React.js är exempel på JavaScript-ramverk som hade kunnat ersätta Angular. Båda dessa ramverk har en komponent-baserad struktur och tar dessutom mindre plats än Angular. Det är preferenserna som avgör vilket som gagnar ändamålet bäst, det hade alltså troligtvis fungerat att lösa Pixelants problem med flera andra JavaScript-ramverk. Det viktigaste verkar vara att bestämma sig för ett ramverk och hålla sig till det. Att lära sig ett nytt JavaScript-ramverk är en tidskrävande process som i slutändan inte nödvändigtvis ger önskade resultat vad gäller funktionalitet och effektivitet.

Det hade varit fördelaktigt att jämföra dessa ramverk mer noggrant innan utvecklingsprocessen. I en sådan jämförelse skulle det då kunnat bestämmas

huruvida ett annat ramverk hade fungerat bättre. Utmaningen i att analysera JavaScript-ramverk i ett tidigt skede är det faktum att svårigheterna i den kommande utvecklingsfasen är okända. Därför kan man dra slutsatsen att det är fördelaktigt att göra en noggrann förstudie inför val av JavaScript-ramverk.

5.4 Versionshantering för ett storskaligt projekt

Sedan uppstarten av själva prototypen har GitHub och språket git används. Det har inneburit code reviews emellan de som varit delaktiga i arbetet, vilket inte endast ger insikt i vad andra projektmedlemmar bidragit med utan också innebär att förändringar i master-branch får en extra granskning. Denna granskning sker utifrån ett nytt perspektiv av en annan utvecklare, i detta fallet Mattias Nilsson med assistans av andra anställda på Pixelant AB. Ofta innebar granskningen en bra förståelse för nya sätt att omstrukturera kod eller enklare misstag som enstaka kommentarer i koden som skulle tas bort.

Utöver code reviews innebar användningen av GitHub att alla nya tillägg av kod kunde följas upp med hjälp av commits. Dessa commits skapades i egna grenar som togs bort då en funktionalitet var färdigimplementerad. Eftersom en specifik funktionalitet tillhörde en särskild gren kunde denna tas bort helt om det visade sig att tillvägagångssättet att utveckla funktionaliteten var felaktigt. Då kunde en ny fungerande gren av huvudgrenen, *master-branch*, skapas och göras om. Slutsatsen som kunde dras var att versionshanteringens inverkan resulterade i en fungerande arbetsmetod. Detta skedde i synnerhet vid utveckling där mer än en person var inblandad.

5.5 Reflektion över etiska aspekter

Följande avsnitt beskriver prototypens sekretess och samhällsnytta.

5.5.1 Sekretess

Eftersom prototypen så småningom kommer innehålla data som tillhör Pixelant är det viktigt att denna information inte är tillgänglig för andra än för dem. Därför har mycket fokus lagts på att se till att endast företagets anställda kan logga in på applikationen för att ändra, visa, ta bort och lägga till data.

Utöver företagets egna behov av att data inte ska kunna kommas åt, så innehåller prototypen också information om deras kunder. Om någon skulle kunna komma åt informationen finns en risk att deras hemsidor kan modifieras. Det här är alltså ett risktagande och som åtgärd för att minimera risken har mycket av utvecklandet inneburit att tillse att prototypen är säker. Det betydde att tokens som slutar gälla efter en viss tid lades till, samt att säkra förfrågningar görs till de API:er som används.

5.5.2 Samhällsnytta

Företaget har ett privat repository som innebär att källkoden bara visas för de som arbetar på Pixelant. Därmed tillåts inte andra företag än Pixelant använda prototypen eller dess koncept, trots att det kan finnas många andra företag som saknar just en sådan tjänst. Det innebär att det prototypen inte bidrar till någon samhällsnytta, utan endast gagnar företaget Pixelant och dess kunder.

5.6 Framtida utvecklingsmöjligheter

Det finns mycket i prototypen som saknas och med fördel hade kunnat läggas till. För tillfället kommunicerar applikationen endast med företagets egna API:er. Givetvis var förhoppningen att kunna lägga till så mycket funktionalitet som möjligt för att förenkla för Pixelants anställda, men den önskvärda mängden funktionalitet innebar mer än vad som fanns inom tidsramen för tiden som spenderats på utveckling.

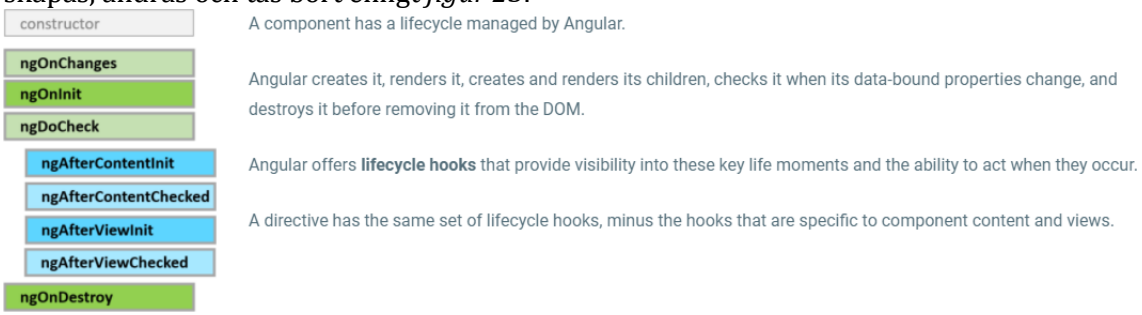
Förhoppningen från början var att skapa samlade API-förfrågningar som skulle skötas med hjälp av GraphQL. Där skulle information från Wrike och liknande tjänster med API-tjänst sammanställas för att senare presenteras i prototypen. Detta är ett utvecklingsområde utanför de tidsramar som examensarbetet gav.

I en fortsatt studie eller genom arbete hos Pixelant hade det varit intressant att undersöka möjligheterna att utveckla prototypen som en native mobil-applikation för Android eller IOS. Eftersom det går att skriva om Angular-applikationer med hjälp av till exempel Ionic finns denna möjlighet. Det är något som diskuterades under utvecklingen på företaget. Prototypen är skalbar och fungerar i en webbläsare för telefonen så som den ser ut nu, men mer funktionalitet från telefonen hade då kunnat implementeras.

I framtiden kommer Pixelant AB att använda sig utav prototypen och vidareutveckla funktionalitet. Det kommer bland annat innebära fler API-kopplingar och komponenter som hanterar dessa.

6 Terminologi

- Angular
  JavaScript-ramverket som använts genom hela examensarbetet.
- Användaren
  En framtida användare av prototypen.
- API
  Application Programming Interface, en beskrivning för hur applikationer kommunicerar med en specifik programvara. [23]
- Augury
  Ett Chrome-tillägg för att visa Angular-komponenters data och tillstånd.
- Back-end
  Databaslösningen från vilken data kring servrar, installationer och användare hämtas.
- Console-fönstret
  Developer tools som finns i alla webbläsare genom att trycka på F12-knappen. Där visas felmeddelanden och även tillägget Augury.
- Företaget
  Pixelant AB där examensarbetet har utförts.
- Lifecycle hooks
  Används i Angular för att bestämma i vilken ordning och hur en komponent skapas, ändras och tas bort enligt *figur 23*.



Figur 23. Visuell beskrivning för hur life cycle hooks fungerar i Angular [6]

- Localstorage
  Data som finns sparad i webbläsaren, data kan exempelvis vara tokens.
- Merge conflict
  En git-term som innebär att koden från den egna grenen inte automatiskt kan sammanslås med master-grenen.
- Native
  Ett system som utvecklats för en specifik plattform eller enhet.
- Prototypen
  Den prototyp i form av applikation som utvecklas med hjälp av Angular 4 i samarbete med företaget Pixelant.
- Pull request
  En funktion i git som innebär en kontroll av kod som godkänns av andra projektmedlemmar av samma git-repository.
- Repository
  Ett projekt skapat med hjälp av git som kollaboratörer kan medverka i.

- Template
  HTML-dokument som renderar design och därmed den del av en komponent som visas för användaren.
- Token
  En autentiseringsnyckel som bekräftas från back-end för att bekräfta att användaren ska få tillgång till olika delar av webbapplikationen. Används av säkerhetsmässiga skäl.
- Wrike
  Verktyg som används för att hantera projekt, user cases och tasks enligt scrum-modellen. Ett verktyg som även påminner om Trello.

7 Källförteckning

I detta kapitel finns de källor som använts under examensarbetet.

7.1 Angular, Web Performance Best Practices

[1] About
https://angular.io/
[Access: 2017-04-03]

[2] Forms
https://angular.io/docs/ts/latest/guide/forms.html
[Access: 2017-03-10]

[3] Component Decorator
https://angular.io/docs/ts/latest/api/core/index/Component-decorator.html
[Access: 2017-05-05]

[4] Services
https://angular.io/docs/ts/latest/tutorial/toh-pt4.html
[Access: 2017-03-10]

[5] Change log
https://angular.io/docs/ts/latest/guide/change-log.html
[Access: 2017-04-18]

[6] Lifecycle hooks
https://angular.io/docs/ts/latest/guide/lifecycle-hooks.html
[Access: 2017-03-10]

[7] Build a master/detail page with a list of heroes
https://angular.io/docs/ts/latest/tutorial/toh-pt2.html
[Access: 2017-06-08]

[8] A quick look at Angular basics
https://angular.io/docs/ts/latest/quickstart.html
[Access: 2017-06-08]

[9] Tutorial: Tour of Heroes
https://angular.io/docs/ts/latest/tutorial/
[Access: 2017-06-08]

[10] Routing
https://angular.io/docs/ts/latest/tutorial/toh-pt5.html
[Access: 2017-06-08]

7.2 Övriga referenser

I följande avsnitt anges de källor som inte är inhämtade från Angulars dokumentation.

7.2.1 Ionic

[10] Tutorial
https://ionicframework.com/docs/intro/tutorial/
[Access: 2017-03-01]

7.2.2 NativeScript

[11] Building Apps with NativeScript and Angular
http://docs.nativescript.org/angular/tutorial/ng-chapter-0
[Access: 2017-03-01]

7.2.3 Android Developers

[12] Manifest
https://developer.android.com/guide/topics/manifest/manifest-intro.html
[Access: 2017-05-05]

7.2.4 Scotch.io

[13] The next version of Angular is 4
https://scotch.io/bar-talk/the-next-version-of-angular-is-angular-v4
[Access: 2017-05-01]

7.2.5 Angular.js blogspot

[14] Ok, let me explain how it's going to be
http://angularjs.blogspot.se/2016/12/ok-let-me-explain-its-going-to-be.html
[Access: 2017-05-01]

7.2.6 Bootstrap

[15] Getting started
https://v4-alpha.getbootstrap.com/getting-started/introduction/
[Access: 2017-05-05]

7.2.7 CSS

[16] Documentation
https://www.w3schools.com/css/
[Access: 2017-05-05]

7.2.8 SASS
[17] SASS basics
http://sass-lang.com/guide
[Access: 2017-06-05]

7.2.9 NPM
[18] Documentation
https://docs.npmjs.com/getting-started/what-is-npm
[Access: 2017-05-05]

7.2.10 GitHub
[19] GitHub Guides
https://guides.github.com/
[Access: 2017-05-05]

7.2.11 Karma
[20] How it works
https://karma-runner.github.io/1.0/intro/how-it-works.html
[Access: 2017-05-10]

7.2.12 Digital Ocean
[21] How to set up ssh keys
https://www.digitalocean.com/community/tutorials/how-to-set-up-ssh-keys--2
[Access: 2017-02-10]

7.2.13 Microsoft
[22] Regular Expression Language - Quick Reference
https://docs.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference
[Access: 2017-06-08]

7.2.14 Make use of
[23] What are APIs, and how are open APIs changing the internet
http://www.makeuseof.com/tag/api-good-technology-explained/
[Access: 2017-06-08]

7.2.15 Font Awesome
[24] Font Awesome, the iconic font and CSS toolkit
http://fontawesome.io/
[Access: 2017-06-07]

7.2.16 Wrike
[25] Scrum in Wrike: making software development more agile
https://www.wrike.com/blog/scrum-in-wrike-making-software-development-more-agile/
[Access: 2017-06-08]

[26] Tour
https://www.wrike.com/tour/
[Access: 2017-06-08]

7.2.17 Augury
[27] Angular Augury
https://augury.angular.io/
[Access: 2017-06-08]

7.2.18 Vue.js
[28] Components
https://vuejs.org/v2/guide/components.html
[Access: 2017-06-09]

7.2.19 React.js
[29] Components and props
https://facebook.github.io/react/docs/components-and-props.html
[Access: 2017-06-09]

7.2.20 Smartsheet
[30] Agile vs Scrum vs Waterfall vs Kanban
https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban
[Access: 2017-06-20]

Appendix

Följande kapitel innehåller större figurer av prototypen.

7.3 Skärmdumpar

Följande avsnitt innehåller skärmdumpar numrerade med *A* som av utrymmesskäl inte ingår i rapporten.

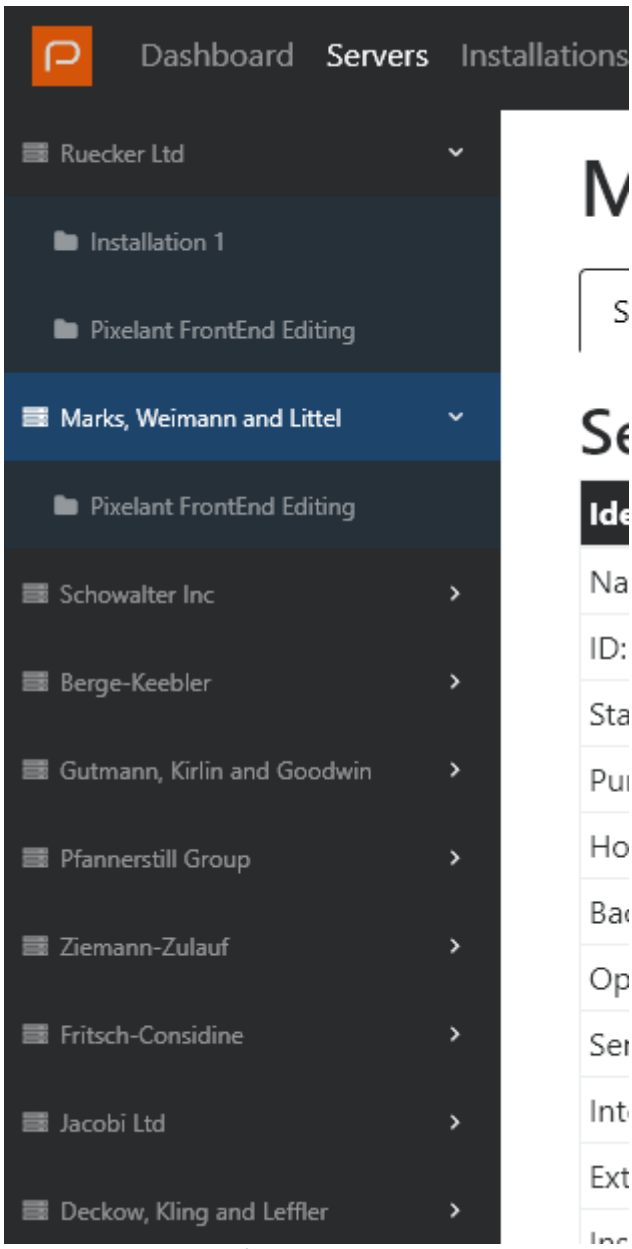


Figure 1A. Sidovy innehållande servrar

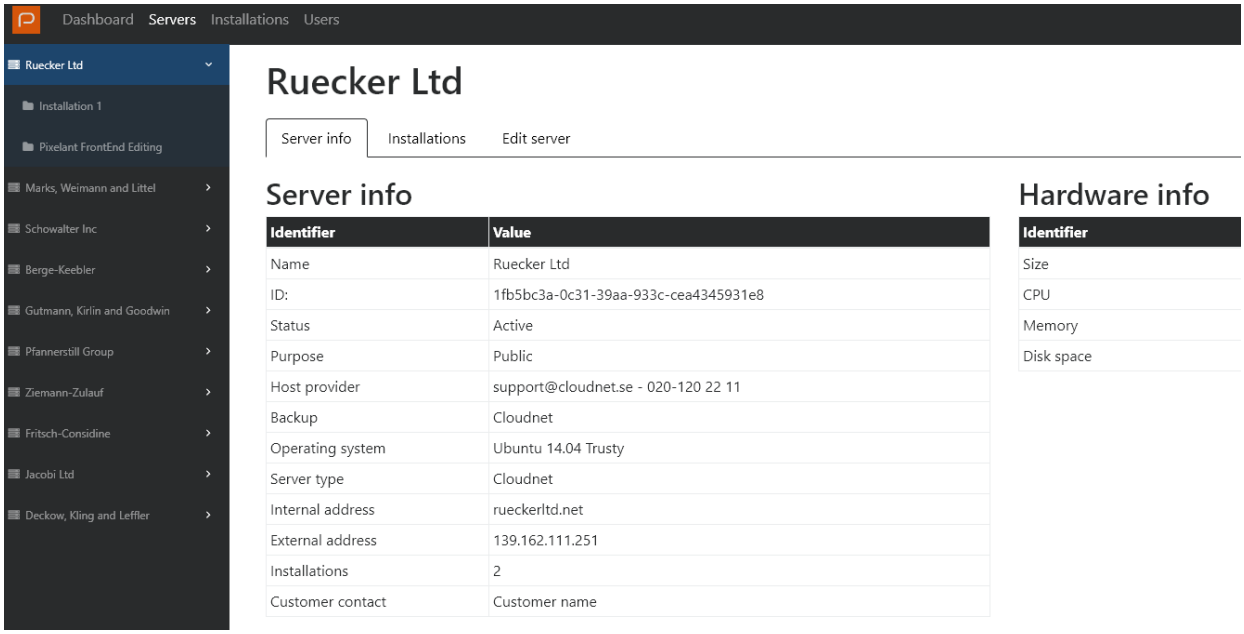


Figure 2A. Servervy för specifik server

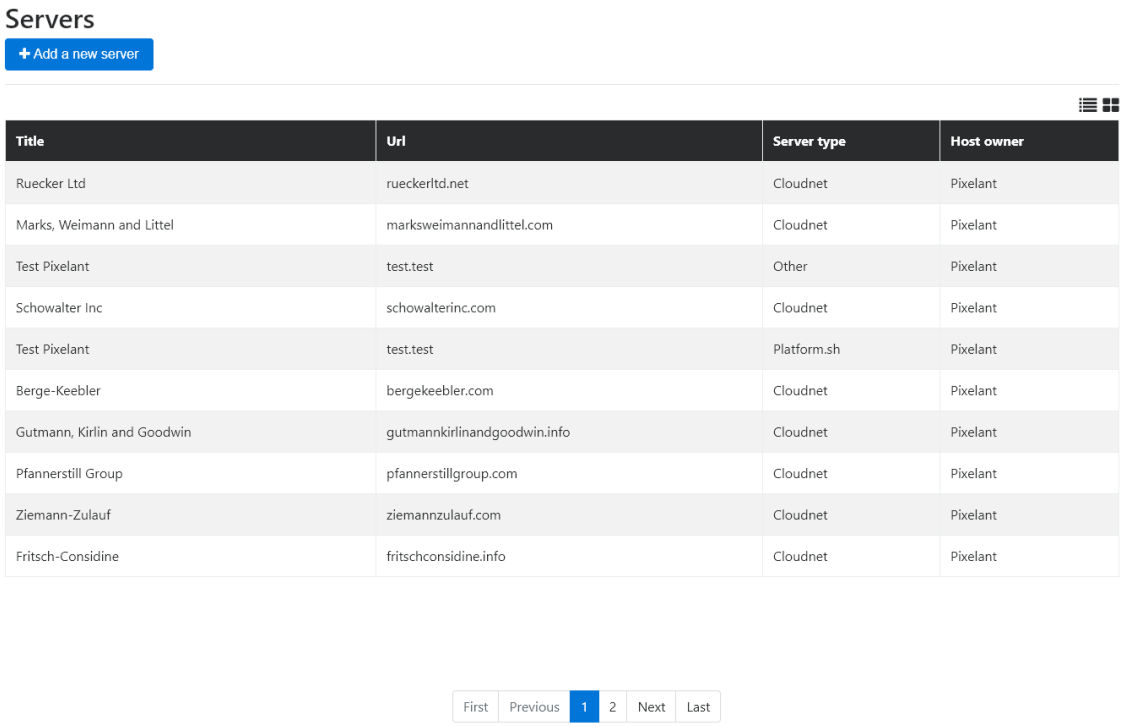


Figure 3A. Listvy för servrar med paginering

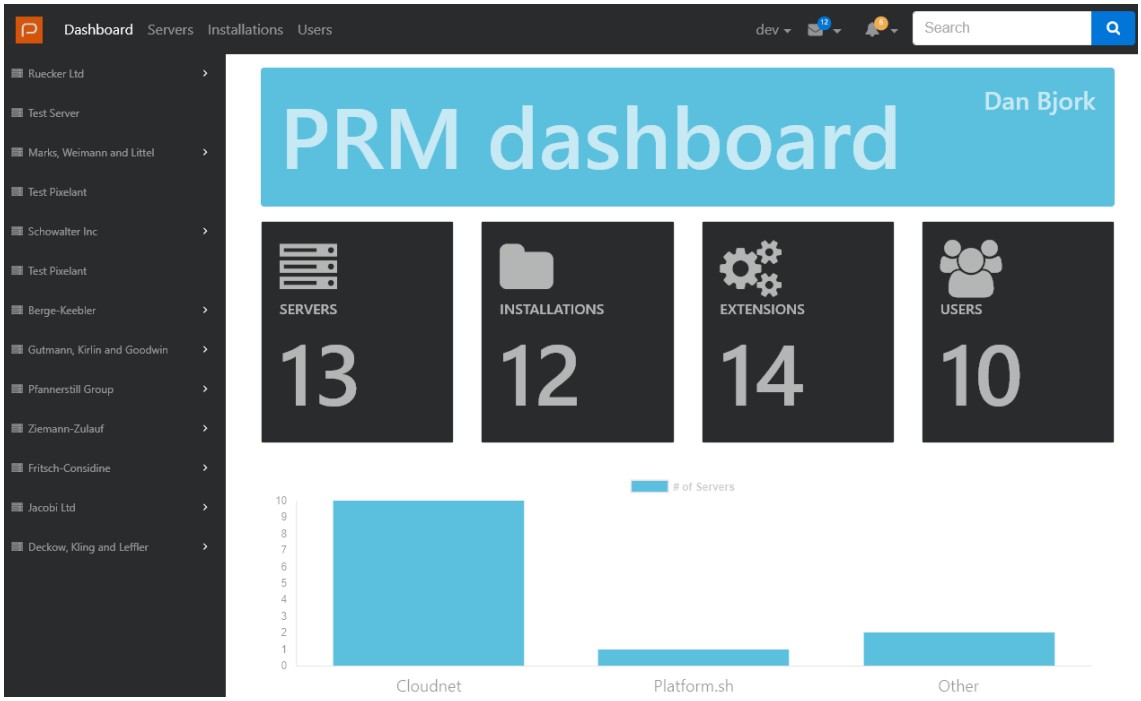


Figure 4A. Vy över prototypens dashboard som visas efter at ten användare loggat in I prototypen