YRKESHÖGSKOLAN
NOVIA

Mall för användarhantering

Jakob Backlund

Examensarbete för ingenjör (YH)-examen

Utbildningsprogrammet för informationsteknik

Vasa 2017

EXAMENSARBETE

Författare: Jakob Backlund
Utbildning och ort: Informationsteknik, Vasa
Handledare: Susanne Österholm

Titel: Mall för användarhantering

Datum 20.4.2017 Sidantal 21

Abstrakt
Syftet med examensarbetet var att skapa en anpassningsbar mall för användarhantering. Mallen skulle senare användas för olika kunder och andra projekt. Inloggning och användarsäkerhet tas inte beaktande i detta projekt, eftersom de är projekt- och kundspecifika. Dessa implementeras senare i de projekt där mallen används. Uppdragsgivaren var eCraft AB.

De tekniker som användes var förbestämda av uppdragsgivaren eftersom mallen skulle kunna användas i befintliga och framtida projekt. De tekniker som användes var: CoffeeScript, Ember.js, MongoDb, Ruby, Bootstrap, HTMLBars och Git.

Utvecklingen gjordes i texthanteraren Atom och planeringen i projekthanteringsprogrammet Trello. Resultatet blev en mall för användarhantering, som i dagsläget är i produktion hos flera av eCrafts kunder.

Språk: svenska Nyckelord: CoffeeScript, Ember.js, Ruby

OPINNÄYTETYÖ

Tekijä: Jakob Backlund
Koulutus ja paikkakunta: Tietotekniikka, Vaasa
Ohjaaja: Susanne Österholm

Nimike: Käyttäjähallinnan malli

Päivämäärä 20.4.2017 Sivumäärä 21

Tiivistelmä

Opinnäytetyön tavoitteena oli luoda muokattavissa oleva malli käyttäjähallintaa varten. Mallipohjaa tullaan käyttämään eri asiakkaille ja projekteille. Kirjautumista ja käyttäjän turvallisuuskysymyksiä ei oteta huomioon tässä projektissa, koska ne ovat asiakas- ja projektikohtaisia. Niitä lisätään myöhemmin uusiin projekteihin missä mallia käytetään. Projektin tilaaja oli eCraft OY.

Jotta mallia voitaisiin käyttää nykyisissä ja tulevissa projekteissa, tilaaja oli jo ennalta päättänyt mitä tekniikoita piti käyttää. Käytetyt tekniikat olivat CoffeeScript, Ember.js, MongoDB, Ruby, Bootstrap, HTMLBars ja Git.

Kehittäminen tehtiin tekstieditori Atomissa ja suunnittelussa käytettiin projektinhallintaohjelmaa Trelloa. Tulos on käyttäjähallintamalli, joka nykytilanteessa on tuotannossa muutamilla eCraftin asiakkailla.

Kieli: ruotsi Avainsanat: CoffeeScript, Ember.js, Ruby

BACHELOR'S THESIS

Author: Jakob Backlund
Degree Program: Information technology
Supervisor: Susanne Österholm

Title: Template for User Management

Date April 20, 2017 Number of pages 21

Abstract
The purpose of this thesis was to create a customizable template for user management. The template is used in different projects by different customers. Login and security are not considered in this project, because they are project and customer specific and implemented in the projects where the template is used.
The employer was eCraft AB.

The tools and programming languages used were chosen by the employer so that the template would be compatible with existing and upcoming projects. Following tools were used:
CoffeeScript, Ember.js, MongoDB, Ruby, Bootstrap, HTMLBars and git.

The template was developed with the text editor Atom and the project planning was done with the project management tool Trello.

Language: Swedish Key words: CoffeeScript, Ember.js, Ruby

Innehåll

1. Inledning

Syftet med detta examensarbete var att skapa en mall för användarhantering med teknologier förbestämda av uppdragsgivaren. Mallen skulle vara möjlig att anpassa till olika kunder och av den anledningen skulle mallen vara generisk.

1.1 Uppdragsgivare

Uppdragsgivaren för examensarbetet var eCraft international, som är en del av eCraft OY AB. eCraft OY AB är ett privatägt företag som grundades i Helsingfors 1999 och sysselsätter över 150 personer i dagsläget. eCraft har kontor i Esbo, Malmö och Vasa. Huvudkontoret ligger i Esbo. eCraft erbjuder sina kunder olika sorters lösningar, exempelvis:

Comment [CB1]: examensarbete

- **eCraft Go.**
  Ett IT system som ger företag möjlighet att ha en komplett lösning för att integrera flera system till en plattform.
- **M3 apps.**
  Erbjuder nya lösningar för att ersätta befintliga affärssystem.
- **Customer relationship management (CRM).**
  En helhetslösning där kunderna kan hantera kundrelationer och fakturering samt säkerställa servicekvalitet. eCraft har vidareutvecklat Microsoft Dynamics CRM med smarta tillägg och mobila applikationer för försäljning, marknadsföring och kundservice.
- **Business Insight.**
  Hjälper kunderna att förbättra och effektivera sin verksamhet, sin rapportering och analys.
- **Enterprice Resource Managament (ERP).**
  eCrafts affärssystem sammanför de viktigaste affärsområden som försäljning, lagring och leverans, vilket gör det enkelt att automatisera de dagliga rutinerna.
- **IT-infrastrukture services (infra).**
  Hjälper kunder att välja rätt system och nätverk samt att underhålla dessa system.

2. Uppgift

Tidigare har eCraft hanterat användare på olika sätt i olika projekt. För att spara tid och pengar beslöts det att göra en universell mall för användarhantering som skulle vara lätt att anpassa enligt de behov som finns inom olika projekt.

Comment [CB2]: mall för användarhantering

Avsikten med det här projektet var att skapa en mall för administrering av användare. Mallen skulle vara generisk och användas i många olika applikationer. I mallen skulle man kunna skapa och administrera användare, roller och grupper. Utgående från vilken roll en användare har skulle man sedan kunna begränsa användarens rättigheter i applikationen.

2.1 Krav

Eftersom mallen ska användas i många olika projekt hade eCraft vissa krav på hur mallen skulle utformas och vilka programmeringsspråk som skulle användas. Mallen skulle vara en webbapplikation utvecklad i HTML med hjälp av ramverket Ember.js och programmeringsspråket CoffeScript. Som databas skulle MongoDB användas.

Mallen skulle vara återanvändbar och lätt att använda. Mallen ska hantera användaruppgifter, roller, grupper samt behörigheter. Möjligheten att begränsa vad en användare har rättighet att göra, beroende på användarens behörighet, skulle vara smidigt. Mallen skulle också ha stöd för tillägg av användaruppgifter enligt de behov som finns. För att underlätta andra projekt skulle mallen vara återanvändbar samt lätt att anpassa enligt de olika projektens behov.

3. Tekniker

Det här kapitlet beskriver de tekniker som användes i projektet. Teknikerna var förbestämda av eCraft, och eftersom projektet var en mall som skulle användas i flera olika applikationer måste den använda samma tekniker som de övriga projekten. Teknikerna som används är Ember.js, CoffeeScript, Bootstrap och Ruby. I kapitlen som följer beskrivs dessa närmare.

Comment [CB3]: skulle skapa en mall

Comment [CB4]: kolla hela texten att du skriver dessa rätt, dvs med stor bokstav där det ska vara det.

3.1 Ember.js

Ember skapades från SproutCore, ett tidigare ramverk skapat av Yehuda Katz och Tom Dale. SprouteCore är ett MVC ramverk som tillhandahåller ett robust JavaScript användargränssnittsbibliotek. De flesta utvecklare ansåg att användargränssnittsbibliotek var onödiga och ett sådant lämnades bort ur Ember.js. (Kelonye 2014 s.5).

Comment [CB5]: ett tidigare ramverk

Comment [CB6]: ?? vad är det

Comment [CB7]: vad är ett användargränssnittsbibliotek?

Ember.js har ett brett användnings område. Det passar utmärkt för applikationer som visar dynamiska data och har omfattande användarinteraktioner. Ember.js är ett ramverk med öppen källkod som används för att skapa webbapplikationer. Med Ember.js kan man skapa komplexa ”client-side” webbapplikationer genom användning av allmänna konventioner (Kelonye 2014 s.5).

En Ember.js webbapplikation består huvudsakligen av komponenterna router, route, controller, view, template, store, model och component (Yakhyaev 2014 s. 7). Nedan följer en beskrivning av dessa komponenter.

- **Router**
  Routern koordinerar applikationens tillstånd med webbläsaren. Den har stöd för vanliga funktioner såsom framåt- och bakåtknappar och att länka till applikationen. Beroende av den aktiva URL:en kallar routern på de matchande rutterna som renderar tillhörande templates (Kelonye 2014 s.8).

- **Route**

Ruttens huvudsakliga uppgift är att förse templaten med en modell (Kelonye 2014 s.9).

- **Controller**
  Templates gör förfrågningar mot controllern efter den data som ska visas för användaren. Controllers manipulerar också data från modellen före den visas för användaren. Controllers har en view och en template (Yakhyaev 2014 s. 8).

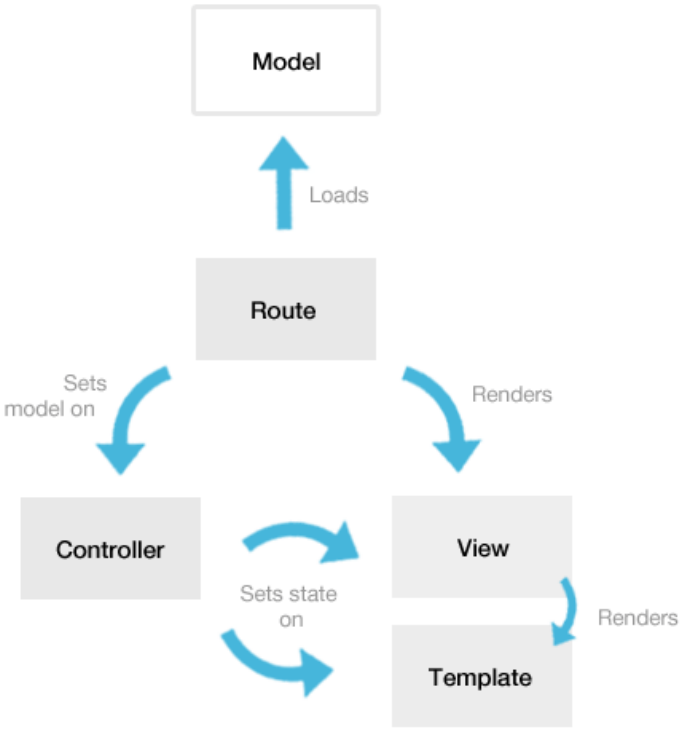
- **Model**
  Modellen definierar strukturen och logiken bakom den data som visas för användaren (Yakhyaev 2014 s. 8).

- **Store**
  I storen sparas data när den har hämtats från servern. Storen har också hand om nya dokument innan de är synkroniserade med servern (Yakhyaev 2014 s. 8).

- **View**
  Views hanterar events. Views delegerar event genererade av användaren till controllern och routen (Kelonye 2014 s.8).

- **Component**
  En komponent är väldigt lik en view och används för att skapa återanvändbara delar till applikationen (Yakhyaev 2014 s. 8).

- **Template**
  Template är mallen som visas för användaren. Componenter, views och controllers har egna templates (Yakhyaev 2014 s. 8).

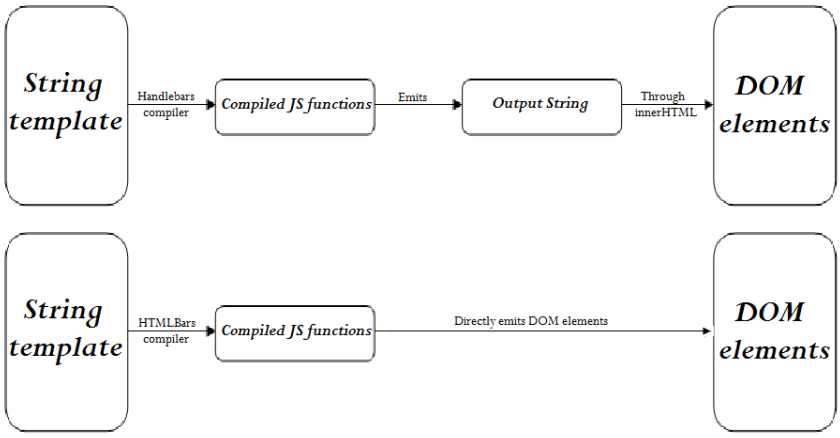


Figur 1: Hur Ember mvc fungerar

(Kelonye 2014 s.8).

3.2 HTMLBars

Ember.js använder sig av HTMLBars, som är ett templatebibliotek som bygger på handlebars. Handlebars och Ember.js är båda utvecklade av Yehuda Katz. Tidigare versioner av Ember.js's handlebars använde sig av textsträng sammanlänkning för att generera vyer. Istället för att använda textsträng sammanlänkning generar HTMLBars så kallade DOM fragment. Figurerna nedan förklarar skillnaden mellan Handlebars och HTMLBars. (Knebel 2013)



Figur 2: Skillnaden mellan handlebars and HTMLBars rendering.

(Knebel 2013)

I detta projekt används HTMLBars för skapandet av användare gränssnittet. Figur 3 demonstrerar användningen av HTMLBars ”each helper” i kombination med ”render helper”.

```
<div class="panel-body">
  {{#each model as |role|}}
    {{render 'admin.user.user-roles.user-role' role}}<br>
  {{/each}}
</div>
```

Figur 3: Användningen av each och render.

3.3 Bootstrap

Bootstrap är ett CSS-ramverk med öppen källkod som ger utvecklaren en mängd lättanvändbara användargränssnittskomponenter, layouts och jQuery-tillägg, såsom navigation, knappar och tabeller. Med Bootstrap kan man enkelt utveckla både för desktop och mobila enheter. Som med alla existerande ramverk så finns det både fördelar och nackdelar med att använda Bootstrap.

Fördelar med att använda Bootstrap är att det fungerar på alla nyare webbläsare, både desktop och mobila. Bootstrap fungerar också på äldre webbläsare exempelvis Internet Explorer 8, men då ser användargränssnittet annorlunda ut än om man har en nyare

Comment [CB8]: det här med stor bokstav... Internet Explorer :D

webbläsare. Vidare är Bootstrap även lätt att anpassa enligt egna behov eftersom Bootstrap är skriven med Less istället för vanlig CSS. Less är ett dynamiskt stylesheet-språk som kompileras till CSS, vilket ger stor flexibilitet. För att få största möjliga nytta av detta bör man använda Less för stilen i webbapplikationen (Niska 2014 s.6).

De största nackdelarna med att använda Bootstrap är att jQuery-tilläggen är svåra att anpassa enligt egna behov. Det anses även att dessa plugin inte är skrivna enligt industrinormer, vilket gör att det kan vara utmanande att jobba med källkoden. Många applikationer som skrivs med Bootstrap ser likadana ut. Detta går dock att undvika genom användningen av egna anpassade teman (Niska 2014 s.7).

3.4 MongoDB

MongoDB är en dokumentdatabas och inte en relationsdatabas. En dokumentdatabas ersätter radmodellen med en mera flexibel dokumentmodell. En dokumentdatabas tillåter inbäddade dokument och Arrays, vilket gör det möjligt att spara komplexa hierarkiska förhållanden i ett dokument. En dokumentdatabas har heller inga förbestämda scheman – ett dokuments nycklar och värden är inte bestämda. Detta gör det mycket enkelt att lägga till eller ta bort fält varefter det behövs (Chodorow 2013 s.3).

Ett av kraven för projektet var att det skulle spara all data i MongoDB. Det här gör det möjligt att spara komplexa dokument. Exempel på dokument som innehåller är användare dokument:

```
{
    "_id" : ObjectId("5870b95030046c5e48a97d57"),
    "firstName" : "John",
    "lastName" : "Doe",
    "username" : "jdoe",
    "email" : "john.doe@foo.com",
    "phone" : "0420301233",
    "isEnabled" : true,
    "roles" : [
        "53f32d1136d6919314ec1c59",
        "53f1e98e4d8c668142f3613c"
    ],
    "groups" : [
        "5870ba2630046c5e48a97d59"
    ]
}
```

Figur 3: Exempeldokument ur user-datasamlingen.

3.5 Ruby

Ruby är ett interpreterande programmeringsspråk, alltså källkoden kompileras då koden körs. Ruby har många likheter med andra programmeringsspråk exempelvis Perl och Python. Rubys objektorientering skiljer sig från dessa programmeringsspråk och är i många avseenden mera lik andra objektorienterade programmeringsspråk (Collingbourne 2008 s 5).

I detta projekt används Ruby för att hämta och spara data. Exempel på hämtning av roller från mongo med Ruby:

```
get do |_request|
  mongo('roles') do |roles|
    roles.find
  end
end
```

Figur 4: Exempel på hur Ruby används för att hämta data.

3.6 CoffeeScript

CoffeeScript är ett programmeringsspråk som kompileras till JavaScript. När man utvecklar i CoffeeScript kan man använda alla existerande JavaScript bibliotek eftersom CoffeeScript i grund och botten är JavaScript. CoffeeScript eftersträvar att göra koden mera läsbar än vad JavaScript är. (MacCaw 2012 s5.) Figur 5 visar skillnaden mellan dessa.

JavaScript:

```
foods = ['broccoli', 'spinach', 'chocolate'];

for (l = 0, len2 = foods.length; l < len2; l++) {
  food = foods[l];
  if (food !== 'chocolate') {
    eat(food);
  }
}
```

CoffeeScript:

```
foods = ['broccoli', 'spinach', 'chocolate']
eat food for food in foods when food isnt 'chocolate'
```

(Coffeescript,u.å.)

Figur 5: Skillnaden mellan en for loop i JavaScript och CoffeeScript.

Exemplen ovan beskriver skillnaden mellan en for loop i JavaScript och CoffeeScript. I JavaScript är kommandona mer svårläsliga än i CoffeeScript.

3.7 Git

Som versionshanteringssystem för detta projekt användes Git. Versionshantering är ett sätt att spara ändringar till filer så att man kan återställa filer och föra historik över ändringar till filer. Git hanterar data som om den vore ett snapshot av ett miniatyr filsystem. Varje gång en ändring görs sparas projektets tillstånd i Git. Git tar i praktiken en bild av hur filerna såg ut vid den stunden och sparar en referens till snapshotet. För att effektivera processen sparar Git inte filer som inte har ändrats sedan senaste snapshotet togs utan sparar en länk till den senaste identiska filen. Git behandlar sin data som en ström av snapshots något som gör att Git skiljer sig från nästan alla andra versionshanteringssystem (Chacon 2009 s 32).

3.8 Atom

Atom är en texteditor med öppen källkod. Atom är utvecklat av Github och beskrivs som en modern skräddarsybar texteditor. Atom är utvecklat med webteknologierna HTML, JavaScript, CSS och Node.Js och körs på Electron. Electron är ett ramverk för att bygga krossplattformapplikationer med webteknologier (Atom, u.å.)

4. Utförande

Kapitlet beskriver hur användarhanteringsmallen gjordes. Först beskrivs planeringen, sedan utvecklingen av de olika komponenterna och sist ut är resultatet.

4.1 Planering

Planeringsfasen inleddes med skapandet av "user stories". Samtidigt som "user storyn" skapades gjordes också en grov teknisk design av projektet. En "user story" beskriver funktionaliteten på ett sådant sätt så det ger värde för användaren eller kunden. "User storys" bygger huvudsakligen på tre aspekter:

- En nerskriven beskrivning av storyn som används som en planerings påminnelse.
- Diskussion om storyn vars syfte är att få fram detaljerna kring storyn.
- Tester som kan framhäva och dokumentera detaljer, vilka kan användas för att fastställa när storyn är klar.

En "user story" kan exempelvis se ut enligt följande: *As an administrator I want to se a list of users.* Detta kunde vara första "storyn" för användarhanteringen. När "storyn" är skapad diskuterar man kring den och antecknar detaljerna på kortet. (Cohn 2009 s4)

Som grund för user storyn hade vi "mockups" som den grafiska avdelningen hade skapat. Det planeringsverktyg som användes var Trello. Det är ett online projektplaneringsverktyg som sparar "user story" kort och detaljer kring korten.

Comment [CB9]: visa exempel

4.2 Utveckling

Comment [CB10]: utan att känna till Ember mera förstår man inte detta

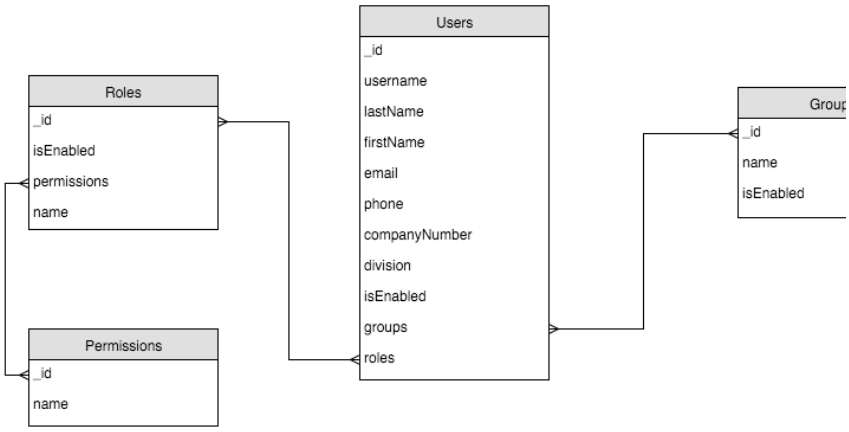
Utvecklingen av applikationen gjordes i textredigeraren Atom. Användargränssnittet utvecklades med Ember.js, HTMLBars och CoffeeScript. Hämtningen och lagringen av data gjordes med Ruby.

Comment [CB11]: redigeraren

Comment [CB12]: inte beskrivet någon stanns

Första steget i utvecklingen var att designa rutterna. Rutterna som skulle ingå var, users, user, roles, groups. När rutterna var planerade designades en domän för att kunna skapa de modeller som behövdes. Figur 6 visar detta:

Comment [CB13]: referera till figuren i texten



Figur 6: Domänmodell över rutterna.

En användare kan ha flera roller och tillhöra flera grupper. Roles och Groups är arrays med id till tabellerna och grupp. Roller kan ha en eller flera rättigheter. Rättigheterna är en array med id till rättighetstabellen. Denna domändesign resulterade i fyra modeller; user, group, role, pemission.

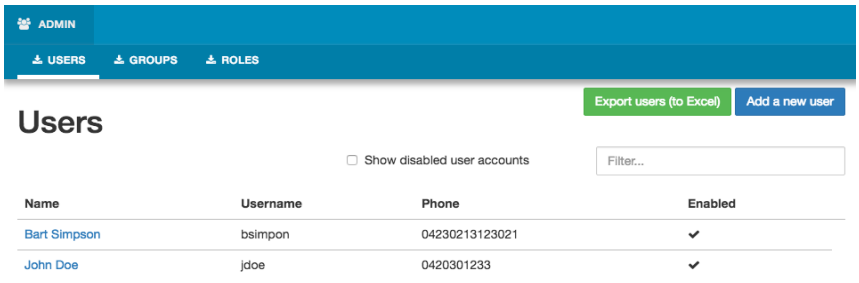
4.2.1 Användare

För att hitta den användare man vill redigera behövs det en listning av användare. I denna vy kan man även filtrera användare och skapa en ny användare. Genom att klicka på en användares namn navigerar man till användarinformations vyn. Där kan en administratör sedan redigera en användares information. Figur 7 visar hur detta ser ut i applikation.

Comment [CB14]: redgera. kolla i hela

Comment [Jb15R14]:

Comment [Jb16R14]:



Figur 7: Lista på användare.

I listan på användare visas information om användaren. Administratören får även en snabb överblick över vilka användare som är aktiverade genom kryssrutan som indikerar om användaren är aktiverad. Listan kan också filtreras enligt användarnamn, email, telefonnummer och för- och efternamn. Genom att klicka på en användares namn navigerar man till användarinformationsvyn.

Filtreringen av en datasamling, i detta fall användare, ansågs vara något som behövdes i många applikationer. För att kunna återanvända filterfunktionaliteten skapades en komponent för filtrering. Filtret är uppbyggt så att man skickar en samling, vilka fält som datasamlingen ska filtreras enligt, och en variabel dit den filtrerade kollektionen sparas.

Template för filterkomponenten är en Ember "input helper". När värdet ändrar så anropas en filterfunktion i controllern för komponenten. För att undvika att listan filtreras innan man ha har slutat skriva används Embers "debounce" metod.

En checkikon används för att visa om en användare är aktiverad. Eftersom denna applikation använder sig av HTMLBars var det enkelt att dynamiskt visa en checkikon om användaren var aktiverad. Htmlabars beskrivs utförligare i kapitel 3.2. Figur 8 visar ett exempel på hur koden kan se ut:

Comment [CB17]: här morrar tanten också

Comment [CB18]: ???

Comment [CB19]: var?

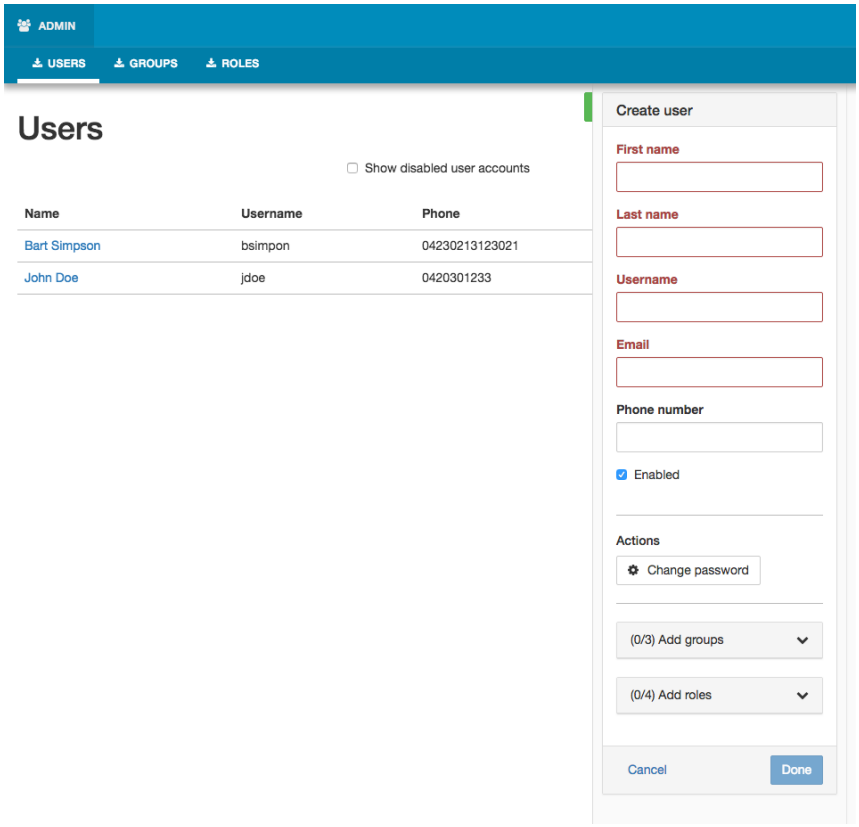
```
<td>
  <i class="fa fa-check {{unless user.isEnabled 'hidden'}}"></i>
</td>
```

Figur 8. Användningen av dynamiska klasser med HTMLBars.

Ikonen som visas kommer från ”font awsome”. Font awsome är ett ikonbibliotek med öppen källkod som baserar sig på CSS och Less.

När man har hittat den användaren som man vill administrera klickar på användarens namn. Det tar en till vyn för användaruppgifter. Från en användare i listningsvyn kan man skapa nya användare genom att klicka på ”skapa ny användare”. Då kommer det fram ett formulär för att skapa nya användare.

Comment [CB20]: visa



Figur 9: Skapandet av ny användare

Comment [CB21]: Ska vara ex. Figur 24. Skapandet av ny användare. Du vill berätta att det är figur 24. Pungt. Sedan vad den visar.

När användaren klickar på *skapa ny användare* skapas en ny "user model" som skickas med vid transitionen till users/edit-rutten. Modellen för edit-mallen blir sedan den nya användarmodellen. Eftersom skapandet av ny användare och editering av ny användare var två väldigt lika vyer skapades en komponent som hade ett formulär för editering och skapandet av användare. Detta gjorde det möjligt att återanvända kod istället för att duplicera den. Detta gjorde att innehållet i vyn endast var komponenten.

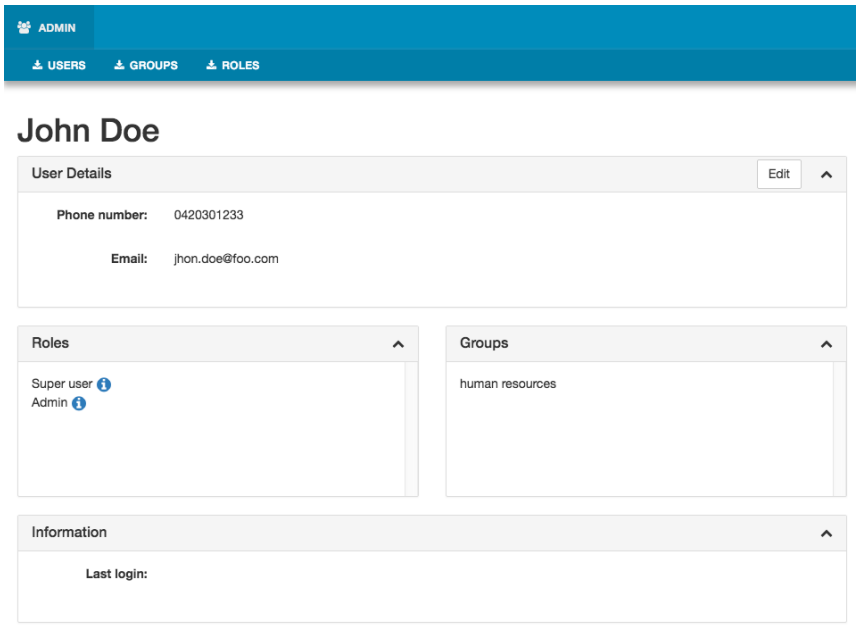
En användare kan ha noll, en eller flera roller och grupper. Rollerna bestämmer sedan vad en användare har tillstånd att göra i applikationen. För att kunna ange en användares roller och grupper behöver man ha data om roller och grupper. Data hämtas i edit-ruttens setupkontroller.

När uppgifterna om användaren är ifyllda görs sedan ett postanrop mot users-resursen där användaren sparas i mongodatabasen, som ser ut enligt följande:

```
post do |request|
  mongo('users') do |users|
    user = users.find_one(
      username: request[:user][:username].to_s.downcase
    )
    fail 'Username must be unique' unless user.nil?
    mapped_user = map_user(request[:user])
    id = users.insert(mapped_user)
    mapped_user[:id] = id.to_s
    mapped_user
  end
end
```

Figur 10: Sparning av användare.

För att få mera info om en användare kan man navigera till användarinformationsvyn genom att klicka på användarens namn. När man har klickat på namnet kommer uppgifterna om användaren fram. Figur 11 visar ett exempel:



Figur 11: Användarinformationsvyn.

Navigeringen sker med hjälp av Embers link to helper. Som modell skickas den användare som man har klickat på. Genom att använda HTMLBars ”each” listas användare och kan skickas med den valda användaren som modell till användarinformationsvyn.

```
{{#each filteredUsers as |user|}}
  <tr>
    <td>
     {{#link-to 'admin.user' user}}
        {{user.firstName}}{{user.lastName}}
     {{/link-to}}
    </td>
    <td>{{user.username}}</td>
    <td>{{user.phone}}</td>
```

Figur 12: Användingen av link to och each.

I användarinformationsvyn ser man användarens kontaktuppgifter samt vilka roller och grupper en användare tillhör. Roller och grupper för användaren hämtas asynkront med hjälp av länkar. Länkarna byggs upp på servern och returneras till klienten.

```
link :roles, '/users/{:_id}/roles'
link :groups, '/users/{:_id}/groups'
id :_id
get do |_request|
  mongo('users') do |users|
    users.find
  end
end
```

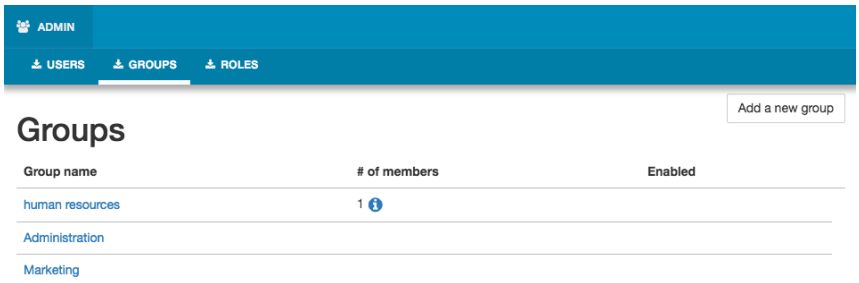
Figur 13: Användningen av linkar för att hämta roller och grupper för användaren.

```
{
  firstName: "John",
  lastName: "Doe",
  username: "jdoe",
  email: "jhon.doe@foo.com",
  phone: "0420301233",
  isEnabled: true,
  id: "5870b95030046c5e48a97d57",
  links: {
    roles: "http://localhost:42000/auth/api/users/5870b95030046c5e48a97d57/roles",
    groups: "http://localhost:42000/auth/api/users/5870b95030046c5e48a97d57/groups"
  }
}
```


Figur 14: Resultat från servern.

4.2.2 Grupper

Användare kan höra till olika grupper och grupperna kan användas på flera olika sätt t.ex. att beskriva vilket kontor en användare hör till. I listan över grupper får man en överblick över vilka grupper som finns.



Figur 15: Lista över grupper.

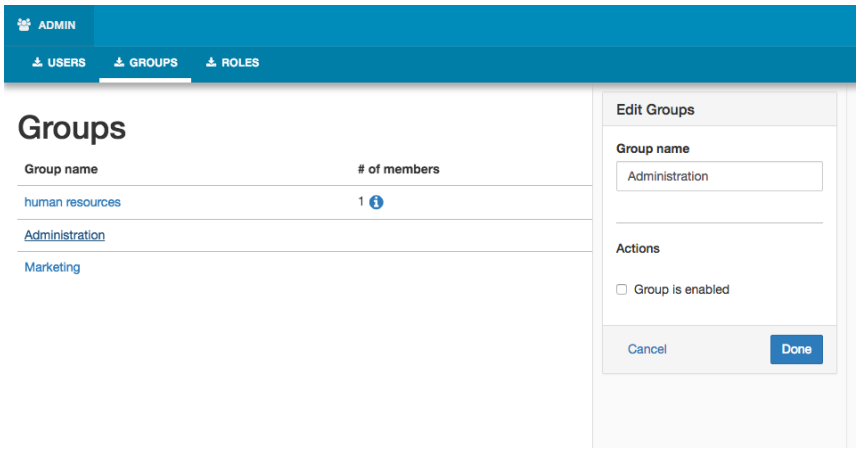
I listan över grupper ser man också hur många användare som hör till en viss grupp. Genom att klicka på infoikonen får man fram en popup med de användare som hör till den gruppen. För att redigera en grupp klickar man på gruppens namn och för att skapa ny grupp klickar man på *add new group*-knappen.

Skapa grupp

För att skapa en ny grupp navigerar man från grupplistningsvyn via *add new group*-knappen. När en användare klickar på *add new group* skapas det en ny gruppmodell som skickas med till edit-routen och blir modell för routen.

Edit routen används både för skapandet av nya grupper samt editering av existerande grupper. När en användare har fyllt i informationen om den nya gruppen och klickat på spara, så görs ett postanrop mot servern och användaren returneras till listvyn.

När användaren returneras till listvyn sker en transiton från edit rutten till groups rutten. När transitionen sker triggrars edit routens resetController hook. I reset-controllern destrueras den ny skapade group-modellen om modellen inte är sparad.



Figur 16 Skapa ny grupp vyn.

Redigera grupp

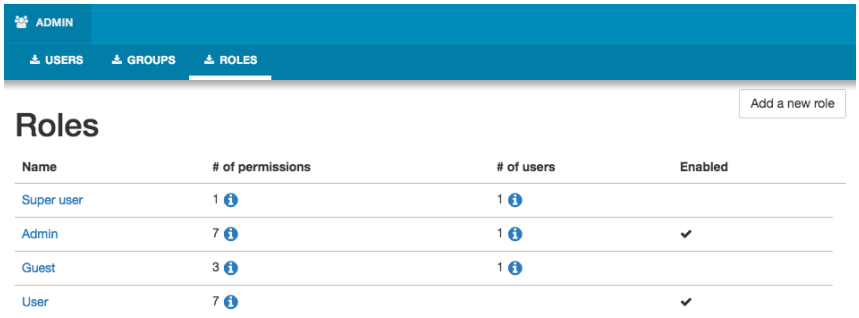
Editering och skapandet av grupper sker i samma route, edit-routen. För att visa skilda texter på knappar och dylikt beroende på om man redigerar eller skapar en grupp användes ember modellens isNew-egenskap. En ember-modell är i nytt tillstånd tills den har sparats.

```
<div class="panel-heading">
   <h3 class="panel-title">
     {{#if model.isNew}}
       {{_NewGroup}}
     {{else}}
       {{_EditGroup}}
     {{/if}}
   </h3>
 </div>
```

Figur 17 Användning av ember-modellens isNew-egenskap.

4.2.3 Roller

För att skapa nya roller navigerar man till rollervyn. I denna vy får man en överblick över vilka roller som redan är skapade och vilka behörigheter och användare rollerna har. Genom att hovra över info ikonerna får man fram en popup som visar vilka behörigheter eller användare som en roll har. Från rollervyn kan man också lägga till eller redigera roller.



Figur 18 Lista av roller.

Skapa en ny roll

För att skapa en ny roll klickar man på *skapa en ny roll*-knappen, vilket öppnar ett formulär där man anger rollens namn samt dess behörigheter. Behörigheterna styr vad en användare har rättighet att göra i applikationen. En roll kan ha flera behörigheter.

När en användare loggar in sparas användarens roller samt behörigheter på session controllern. Genom att använda en HTMLBars if-sats kan man enkelt dölja information som den inloggade användaren inte har rättighet till att se. Figur 19 visar hur:

```
{{#if session.isAuthenticated}}
  {{#altus-navbar
    linkTo="index"
    inverse=true
    helpButtonText=helpButtonText
    showHelpAction="showHelp"
    bottom=false}}
```

Figur 19 Användningen av behörigheter med if-sats.

I mallen finns det ingen möjlighet att underhålla behörigheterna, utan det måste göras av en databasadministratör.

Redigera roller

För att redigera roller klickar man på rollens namn och samma formulär som för att skapa roller visas. Editerings-och skapningsvyn var så pass lika att de kunde använda samma vy. Skapandet och redigerandet av roller sker på samma sätt som skapandet och editerandet av grupper.

5. Resultat

Resultatet av examensarbetet blev en funktionerande template för hantering av användare samt deras behörigheter. Mallen används i produktion hos flera kunder i dagsläget. Användargränssnittet är uppbyggt med hjälp av Bootstrap och klient logiken är byggd med Ember.js och CoffeeScript.

Mallen har inte tagit i beaktande användarsäkerhet eller inloggning, eftersom dessa är projektspecifika och implementeras i de projekt där mallen används.

6. Diskussion

Resultatet blev en simpel template för hantering av användare. Mallen har många förbättringsmöjligheter som vi inte kunde använda på grund av kraven som ställdes på mallen.

Istället för att använda MongoDB som databas kunde man ha använt en relationsdatabas som tex postgreSQL och som backend Ruby on Rails. Detta hade gett backenden mera struktur samt ett bättre underlag att utveckla vidare på.

Front End kunde ha byggts med ember cli vilket skulle ha gett en mer strukturerad frontend samt möjligheter att använda existerande ember cli tillägg. En annan fördel med att använda sig av ember-cli är att man på ett enkelt sätt kan skapa automatiserade tester för koden.

7. Källförteckning

Atom, u.å Hämtat 16.4.2017 från https://atom.io/

Chacon, S och Straub B. 2009. Pro Git. New York. Apress.

Chodorow, K. 2013. *MongoDB: The Definitevie Guide, Second Edition.* Sebastopol: O'Reilly Media, Inc.

CoffeeScript, u.å Hämtat 16.4.2017 från http://coffeescript.org/

Cohn, M. 2009. User Stories Applied for Agile Software Development. Boston: Pearson Education, Inc.

Collingbourne, H. 2008. *The Little Book Of Ruby.* Devon: Dark Neon Ltd.

Kelonye, M. 2014. *Mastering Ember.js*. Birmingham: Packt Publishing.

Knebel, J. 2013. *An In-Depth Introduction To Ember.js.* hämtat 12.2.2017 från https://www.smashingmagazine.com/2013/11/an-in-depth-introduction-to-ember-js/#what_is_handlebars_template_precompiling

MacCaw, A. 2012. The Little Book on CoffeeScript. Sebastopol: O'Reilly

Niska, C. 2014. *Extending Bootstrap.* Birmingham: Packt Publishing.

Yakhyaev, R. 2014. *Ambitious Ember Applications A comprehensive Ember.js tutorial.* Leanpub.