Examensarbete SUW 2015

BILDUPPLADDNINGSAPPLIKATION

Jarl Lindquist
WEBBUTVECKLING AGIL SYSTEMUTVECKLING

Innehåll

Sammanfattning

Detta arbete handlar om databaskopplingar samt hur arbetet med att bygga upp denna webbapplikation har sett ut.

Arbetet visar att för den som har lärt sig ett visst programmeringsspråk är den det bästa i alla fall rent tidsmässigt att arbeta med om man ska göra mindre webbapplikationer. Angularjs håller fortfarande att arbeta med när det gäller att bygga applikationer även om den har ett antal år på nacken. Däremot om man ska göra något i framtiden och då speciellt inte har någon större tidsaspekt är de nyare ramverken bättre att arbeta med eftersom de är bättre utvecklade för dagens teknik inom webbutvecklingen.

Gällande databaskopplingarna visar resultatet av arbetets gång att ORM tekniken har underlättat arbetet med att interagera med databasen. Utifrån servermiljön som är NodeJs användes sequelzejs som hjälpmedel. Denna hjälpte till med att hantera modellerna för databasen samt CRUD metoderna för data till och från databasen vilket medför att man inte behöver skriva de mer enklare ”queries” till databasen. Genom att mappa upp modellerna för hur tabellerna ska se ut och heta kan sequilizejs bygga tabellerna i databasen och detta medför att man inte behöver lägga samma tid i databasen för att skapa strukturen eftersom SequlizeJs hanterar detta. Detta är en tidsaspekt som kan tas med i tankarna även om själva designen av databasen ska vara klar innan man börjar med den även om man väljer detta sätt.

Inledning

Abstract

Under utbildningen till Systemutvecklare Agil Webbutveckling har fokus legat på att lära sig att arbeta med programmeringsspråket JavaScript på många olika sätt. JavaScript har utvecklats till att bli ett programmeringsspråk som kan användas till både backend (serversidan) och frontend (vad webbläsaren visar). Detta medför att man kan arbeta i samma programmeringsspråk oberoende om man programmerar för backend eller för frontend. För att kunna applicera alla de färdigheter och kunskaper som inhämtats under utbildningen valdes en applikation för att beskriva hur arbetet har utförts.

Arbetet handlar om hur man har byggt en applikation och vilka tankar som uppkom när man har valt bland olika tekniker som finns för att skapa webbapplikationer. Till exempel vilket ramverk som skulle kunna passa till detta arbete eller om man ska gå helt ramlöst och bara skriva det med vanligt JavaScript.

Eftersom man ska kunna ha möjlighet att få tillgång till både koden och arbetet på ett bra och enkelt sätt finns båda delarna på Github med ett repo[1]. Github är en versionshanteringsapplikation där man kan lägga sin kod. Genom olika delar inom Git, för att skapa grenar, får man en bättre hantering på utvecklingen inom både små och större utvecklingsprocesser. Bland annat kan man skapa olika grenar när man arbetar med olika saker och när man är klar sammanfogas grenarna med varandra[2]. Adressen för koden på Github är: https://github.com/webcrunch/examensarbete

I Github skapades det en lista som förklarar hur man gör när man ska sätta upp applikationen. Det finns även en fil(exDB.sql) som innehåller databasstrukturen som går att importera till det databasverktyg som man använder. Det kan vara till exempel phpAdmin eller Heidi sql eller något annat verktyg. Det går även att importera filen om man sitter i en terminal och arbetar.

Även om man inte importerar denna fil är det inga problem eftersom en viss teknik inom applikationen, som vi kommer att återkomma till i texten, löser interaktionen med en tom databas.

För att hantera all viktig information om alla lösenord och annan viktig information skapade jag en fil i mappen modules som hanterar all den informationen. För att Nodejs ska kunna nå detta deklarerades det en variabel som man kan nå all data från.

```
serverConst          = r('./modules/auth.js'),// <--IMPORTANT
```

[1] https://help.github.com/articles/create-a-repo/

[2] https://nvie.com/posts/a-successful-git-branching-model/

Denna filen ser ut så här, all text som kommer efter de dubbla / eller mellan /* */ är kommentarer:

```
var path = require("path");
var appRoot = path.normalize(__dirname + "/../" +  '/');

   module.exports = settings = {
   /*server information */
             endpoint : '*',
             webRoot: 'www',
             appRoot : appRoot,
             fileRoot: appRoot + 'www',
             indexFile: '/index.html',
             port: 3002, // insert the port the server listens to
   /*Amazon information*/
             signatureVersion: 'v4',               //
  secretAccessKey: '', // insert secret Access  Key
  accessKeyId: '', // instert secret Key Id
  region: '', // Witch region
  bucket: '', // the name of the bucket
   acl: '',
   key: '',
  secret: '',
  bucketPath: "",

/*Database information*/
  host: '', // whitch host
 user: '', // whitch user??
 password: '', // whitch password
 database: '', // the name of the database
 protocol: '', // sort of protocoll (used by sequlize)
portServer:  // Witch port to database
};
```

Här finns all blandad information, allt från den information som Nodejs behöver för att hantera webbsidorna till databasens uppsättande.

Samt att den är omdöpt till tempA.js. Detta är gjort för att den information som ska ligga där är anpassad efter olika variabler det vill säga olika användarnamn och lösenord. När man lägger upp personlig information gällande system på GitHub och då speciellt ett öppet repo kan detta medföra en säkerhetsrisk för den personen eftersom alla har tillgång till alla filerna och ser då all hemlig information. Efter att man har hämtat hem projektet ändras namnet på tempA.js till auth.js

Syfte

Syftet har varit att skapa ett ”Content Management System” för att se vilka val man har gjort för att skapa detta CMS. Syftet har också varit att genom de kunskaper som man har inhämtat under utbildningen skapa denna applikation. Men även för att kunna se om man kan undersöka ny funktionalitet till applikationen och genom detta inhämta ny kunskap.

Vad menas med ett CMS då? Utifrån den bild som jag har av ett CMS är det ett system där man kan spara information genom. Den information som har fokuserats mest på är bilder men tanken har varit att fokusera på andra delar också. Till exempel textinformation eller datamängder som siffror. Allt kommer att vara beroende på tid och möjlighet. Applikationens struktur och syfte är att användaren ska kunna visa bilder som de har laddat upp. Andra delar som förhoppningsvis ska kunna finnas med är att ge kommentarer till bilderna och ge dem betyg på ett skalbaserat betygsystem. Applikationen kommer att anpassas till mobila enheter. Detta görs med en teknik som återkommer senare i texten. Däremot kommer fokus att ligga på laptop/dator design.

Frågeställning

Vilka problem kommer då att hanteras i detta arbete?

Den stora problemformuleringen kommer att vara vilka hjälpmedel som man har använt sig av för att bygga applikationen samt vilka fördelar/nackdelar det har varit med att använda sig av dessa.

Hjälpmedel kan vara olika ramverk som finns att tillgå när man utvecklar för webben men det finns även olika sätt att interagera med data till och från databasen.

Vidare kommer man även att analysera arbetssättet. Hur arbetet har utvecklats och vilka problem som har tillkommit.

Hur har arbetet utvecklats?

Metoddelen

I uppsatsen kommer arbetets förlopp att beskrivas i ungefärliga arbets- och tidsaspekter. Iden och målet var att bygga en applikation som möjliggjorde hantering av bilder. För att kunna utveckla denna idé behövde man ett antal byggstenar för att både hantera och visualisera det alster som har utvecklats. Den första byggstenen är en server att kunna hantera webbsidor. Det mesta av applikationens data kommer att hanteras genom servern. Detta gjordes med Nodejs. Det är en server skriven i JavaScript. Nodejs kan förklaras så här:

> *Node.js uses an event-driven, non-blocking I/O model that makes it lightweight and efficient.*[3]

> *In simple terms, it's a JavaScript free and open source cross-platform for server-side programming that allows users to build network applications quickly.* [4]

Detta medför att den enbart har det viktigaste från början och sedan är det upp till utvecklaren att bygga på vidare. Man kan säga att Nodejs är likt ett näst intill tomt byggblock där man kan sätta på mindre byggstenar, som heter moduler, för att skapa en server som kan hantera det man vill att den ska göra. De moduler som man ska använda måste deklareras och det görs bäst i början av server filen som i vårt fall är app.js. Den är uppbyggd i olika delar som i sig är beroende av varandra.

Har valt att skapa en alias för require[5]. Require är en del av Nodejs som förklarar vilken modul som ska vara tillsammans med vilken variabel. Denna är satt till det lilla r i början. Detta gör att man kan skriva r istället för require.

```
const
r = require,
express = r('express'),
fs = r("fs"),
path = r("path"),
multer = r('multer'),
app = express(),
aws = require('aws-sdk'),
multerS3 = require('multer-s3'),
serverConst = r('./modules/auth.js'), // <--IMPORTANT
Session = r('express-session'),
bodyParser = r('body-parser'),
mysql = r('mysql'),
js_orm = r('js-hibernate'),
Sequelize = r('sequelize'),
```

Här ser man hur de modulerna som Nodejs använder sig av.

Vi vill att servern ska skicka de sidor som webbläsaren begär. Dessa block kan man även se som paket eftersom Nodejs har en npm(node package manager)[6] som hanterar alla moduler som kopplats till Nodejs.

[3] https://medium.freecodecamp.org/what-exactly-is-node-js-ae36e97449f5
[4] https://medium.com/@LindaVivah/the-beginners-guide-understanding-node-js-express-js-fundamentals-e15493462be1
[5] https://medium.freecodecamp.org/requiring-modules-in-node-js-everything-you-need-to-know-e7fbd119be8
[6] https://www.npmjs.com/

Express/body-parser

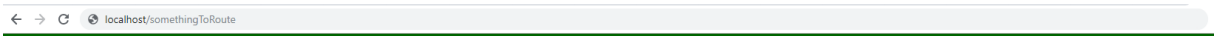
Vår första byggsten som vi använder är Express. Express är ett tredjeparts framework till Nodejs. Den ger en struktur till Nodejs och hjälper till att hantera olika former av http förfrågningar. Det finns många olika http förfrågningar men de som används mest är GET, POST, PUT och DELETE.[7]

GET förfrågan används oftast för att hämta data från servern till webbläsaren. POST används mestadels för att skicka data till servern från webbläsaren. PUT och DELETE är mer använda vid arbete med databasförfrågningar.

It's a web framework that let's you structure a web application to handle multiple different http requests at a specific url[8].

Denna används för att hantera dirigeringen(Routing) av sidorna. Dirigeringen är till exempel när man skriver in olika adresser i webbläsaren.

Dessa hanteras av Express. Beroende på vilken sorts protokoll så kan Node/Express skicka tillbaka olika sidor eller text. Till exempel om man skriver in "localhost/somethingToRoute" i webbläsaren



hanteras denna adress delvis som ett GET anrop. Detta hanteras utifrån http protokollet och Express hjälper till att utföra det man har sagt till servern vad den ska göra när den adressen skickas till serven kommer upp. Express hanterar alla sorters förfrågningar men det har för det mesta varit enbart GET och POST förfrågningar till servern. Express och Nodejs läser av protokollet och utgår ifrån rutten och protokollet. Om vi tar rutten ovanför(localhost/somethingToRoute) så är den av protokollet GET. Nodejs koden som hanterar detta ser ut så här:

```
app.get('/somethingToRoute, (req,res)=>{
  Olika funktioner.

})
```

Om man skickar in information med POST ser det liknande ut fast då blir det bara app.post.

Allt detta kan man se om man analyserar filen app.js. Filen går att finna antingen i den mapp man hämtat hem från github sidan eller på github sidan. Där ser man även att applikationen använder en annan modul som heter body parser. Denna modul hjälper till att hantera den data som kommer in och skickas tillbaka som Json. Den hanterar hela body delen från Express Detta gör det enklare att komma åt data eftersom man kan bygga upp den data som man skickar på ett visst sätt så förstår servern genom body parser var den ska leta efter informationen.

Alla komponenter som ska ingå i applikationen listas i package.json. Filen hanterar alla beroenden(modules) i Nodejs. Den finns att syna i samma mapp som man kan hitta app.js. Applikationen är uppbyggd utifrån ramverken Angularjs, Twitter Bootstrap version4 och jQuery.

Utifrån package.json kan man hämta alla ramverk och verktyg som ska användas på webbsidan(frontend). Detta gör att man kan ladda ned dem istället för att använda sig av " Content Delivery Network" – CDN som hämtar informationen från en annan server. Detta gör att man kan ladda ned alla paketen tillsammans med de andra modulerna samt att man inte behöver vara

[7] https://medium.com/@LindaVivah/the-beginners-guide-understanding-node-js-express-js-fundamentals-e15493462be1

beroende av internet för att nå dem eftersom de redan är nedladdade. För att kunna använda dem till webbsidan behöver man skicka dem till webbsidan och det gör man genom Express-ramverket.

” app.use('/js', express.static(__dirname + '/node_modules/bootstrap/dist/js')); // redirect bootstrap JS” – kod app.js line 228”

Denna kod skickar Bootstrap filerna som behövs för att kunna använda ramverket på webbplatsen till webbplatsen. I index.html filen läser man in de filer som blev medskickade. Sökvägen till filerna startar från mappen js i node_module:

“<script src="/js/bootstrap.min.js"></script>” – kod index.html – line 23”

Package.json hanterar dem och första gången som systemet körs i till exempel ett terminalfönster så är inga beroende(dependence) installerade till både servern men även ramverken. Genom att använda sig av kommandot npm install installeras alla beroenden som är listade i package.json.

```
"dependencies": {
    "body-parser": "*",
    "jquery": "^3.2.1",
    "compression": "*",
    "angular": "1.6.6",
    "cookie-parser": "*",
    "bootstrap": "4.0.0-alpha.6",
    "debug": "*",
    "express": "*",
    "express-session": "^1.14.2",
```

Detta är en del av package.jsons lista av de beroenden som är använda till denna applikation. Vissa används bara inom servermiljön alltså i Nodejs medan andra som till exempel både Angular och jQuery laddas ned för att användas till webbplatsen(frontenden).

För att hantera alla bilder och information som användaren vill ha sparade används en databas av typen MySQL. Det finns många olika sorters databasversioner. De mest använda är MySql, NoSql och PostgreSQL. För att servern ska kunna prata med databasen finns en modul som hanterar detta smidigt, nämligen Mysql. Den hanterar all datatransförering till och från databasen.

En annan modul som använts och som arbetar med databasen är Express-session[9].

Den hjälper till att hantera sessioner som startas när en användare loggar in på applikationen. Genom att starta en session går det att se om en användare är inloggad eller ej. Detta möjliggör en koll så att användaren inte behöver logga in varje gång som den surfar till denna webbapplikation.

ORM

För att interagera med databasen användes det ett ORM. ORM står för Object-relational mapping och den är ett objektorienterat system som konverterar databastabeller till klasser. De rader som finns i databasens tabeller konverteras till objekt och cellerna med information blir attribut till objektet.

ORM används till många olika programmeringsspråk. Har funnit att det används ibland annat .Net, Java, Python samt Javascript. Det finns ett antal ORM moduler för JavaScript. Beroende på vilken sorts databas som man använder, när man utvecklar sin applikation, kan man använda olika moduler för ORM tekniken. MongoDb[10] som är en NoSQL databasversion använder sig av ett system som arbetar efter liknande struktur som ORM. ORM arbetar med modeller och i MongDB är det inte lika uppstyrt och då kan man använda sig av Mongoose[11] för att ge modellen striktare regler.

Om man tar MySql versionen så fanns där ett antal andra moduler. Den första som användes under utveckling av denna applikation var js- hibernate[12]. Tyvärr som med många av de andra modulerna saknade den tillräckligt med funktioner som behövdes för denna applikation. Efter lite letande hittades modulen Sequelizejs[13] och den hade fler funktioner och passade till det jag behövde göra till applikationen.

Sequlizejs använder alla funktionerna till databasinteraktion(CRUD). C som står för CREATE hanterar skapande av nya poster i databasen. R som står för READ läser data som redan finns i databasen. U som står för UPDATE uppdaterar data som finns i databasen. Den sista är D som står för DELETE alltså att kunna ta bort data från databasen. Denna funktion har även MongoDB som nämndes ovan.

Exempelvis då man ska hämta något från databasen används både MongoDB och Sequelizejs ”tabellnam.find(parameter)” ändelser med parametrar inom parenteser som finns i databasen. Det kan se ut så här:

User.find({where:{email : changeEmail }})[14]

Detta kan likställas med sqls ”SELECT * eller parameter FROM tabellnamnet. Inom ORM byter man ut sql frågorna mot modell.find, modell.create, modell.update.

[9] https://stormpath.com/blog/everything-you-ever-wanted-to-know-about-node-dot-js-sessions
[10] https://www.mongodb.com/
[11] https://mongoosejs.com/
[12] https://www.npmjs.com/package/js-hibernate
[13] http://docs.sequelizejs.com/
[14] App.js rad 299

ORM modeller

Utifrån filen models.js skapar man alla modeller till tabellerna i databasen. Genom ORM kollas det också om dessa är skapade i databasen eller inte. Det har även används en annan modul som är ORM baserad och det är Hibernatejs. Skillnaderna är små men det som är likadant för de båda är att de behöver skapa en modell över hur databasen ser ut och arbeta utifrån den.

En modell kan se ut så här:

```
Tuser: sequelize.define('tempuser', {

  userName:
   Sequelize.STRING,

  firstName:
   Sequelize.STRING,

  lastName:
   Sequelize.STRING,

  password:
   Sequelize.STRING,

  email:
   Sequelize.STRING,
  userType: Sequelize.STRING
   }
```

Första delen visar hur Sequlizejs bygger upp sina modeller.

Man skapar ett variabelnamn, i detta fall Tuser, för den temporära användarens data innan den har aktiverat sitt konto. För att Sequelizejs men även den andra modellen ska kunna veta vilken tabell den ska skapa eller leta efter definierar man den med det namn man vill att den ska heta eller heter, i det här fallet tempuser. Vidare skapar man de olika kolumnerna och deras parametrar. Till exempel om den data som ska skickas in ska vara text är det string som parameter för den kolumnen. Det går att sätta många olika regler på detta sätt. Utifrån denna uppbyggnad använder man en del av detta när man ska interagera med databasen.

```
 const userMap = session_js.tableMap('users')
.columnMap('id', 'id', {isAutoIncrement: true})
.columnMap('uName','username')
.columnMap('fName','firstName')
.columnMap('lName','lastName')
.columnMap('pass','password')
.columnMap('roll', "userType")
.columnMap('email','email')
.columnMap('updateP','updateP')
.columnMap('date', "time", {isAutoIncrement: true});
```

Denna visar hur Hibernatejs bygger upp sina modeller. Vidare i förklaringen om hur ORM används är det Hibernatejs som visas i kodexemplen. Däremot har det varit Sequlizejs som det legat störst fokus på.

I Node sparar man modellen till en variabel som man ger ett namn:

```
 // Variable that store the model of Temporary Users from moduels -> models.js
const TUser = models.Tuser;
TUser.sync({logging: false});
```

Denna tabell används när en användare ska läggas till efter att den har aktiverat sitt konto. Då skapar man en fråga(query) till databasen:

```
 var query = session_js.query(userMap)
             .where(
        userMap.email.Equal(email) // =
        );
```

query.then(function(result){ ← denna del är till för att hantera svaret från frågan ovanför.

Denna kod är till för att kolla om det redan finns en användare med den emailen eller inte. Denna kod är byggd i Hibernatejs och detta medför att den ser lite annorlunda ut än den kod som är skriven i Sequlizejs, men de fungerar likadant. Hibernate tar modellen av tabellen. Modellen är userMap som är till tabellen users. Den har även en parameter som den ska leta efter när den ska hämta information från tabellen och det är att den bara ska ta med information som matchar den data som finns i kolumnen email. Detta innebär att om det finns data i email kolumnen tas detta med annars blir det ett tomt objekt tillbaka. För att göra samma sak med Sequlizejs hade man skrivit som det är beskrivet längre upp i texten.

User.find({where:{email : changeEmail }})

Detta visar på två olika tänk inom samma struktur för de båda modulerna. Eftersom Hibernate samt Sequlizejs är asynkrona använder man sig av then för att få en ”promisse”[15] att information ska komma. Genom detta kan man skapa en funktion som körs när informationen som lovats kommer. Och informationen är i detta fallet i variabeln result.

Applikationens uppbyggnad

Applikationen är uppbyggd med Angularjs som är ett ramverk för SPA[16] (Single Page Application) tänk.

SPA tekniken medför att den inte läser in hela sidan hela tiden utan den byter ut den vy som man ber om. Angularjs är ett MVC (Model, View, Controller) ramverk som arbetar med att särskilja de olika uppgifterna mellan varandra. Det enda som användaren ser är vyn. Den hanterar alla designmässiga delar. Controllern hanterar all data som skickas och hämtas från modellen. Modellen är den lägsta nivån och det är till exempel i den delen i Angular som $scope hanteras.

[15] https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then
[16] https://www.codeproject.com/Articles/1224654/Single-Page-Application-using-AngularJs-Tutorial

För att kunna arbeta med en viss design, även om fokus oftast låg på att det skulle fungera rent funktionsmässigt, så användes både CSS och ett ramverk som hjälper till att utveckla CSS:en och det är Bootstrap. Applikationen är byggd med Bootstrap 4 som är en uppdatering från Bootstrap 3. [17]

Har valt att göra designen efter en mer spartansk utformning med en huvudmeny i toppen och en sidfot som visar statisk information. Mittdelen är föränderlig beroende på vilken sida man är inne på. Det vill säga vilken länk som man valt att följa i huvudmenyn

Applikationen har ett antal knappar som används för att interagera med. Det finns hem-knappen, titta-på bilder som är uppladdade knappen, administrations-knapp. Denna knapp visas bara om man är inloggad i systemet. Den sista knappen är till för att skapa ett konto till systemet. Alla knapparna syns i huvudmenyn.

Genomgång av applikationen

Hela idén med applikationen är att kunna ladda upp bilder. För att kunna göra detta behövs ett konto för användaren. Registreringen fungerar på det sättet att användaren skriver in användarnamn, email och lösenord i ett formulär. Dessa parametrarna skickas till servern som bland annat gör om lösenordet till oläsliga tecken. I detta objekt som skickas in läggs det även på ett id. Id:et är en markör för att det lättare ska kunna sökas efter denna information. Informationen läggs till på en temporär plats i en av tabellerna i databasen. Samtidigt som detta händer skickas det ut ett verifieringsmail till den e-mail som användaren valde att registrera sig med. I detta mail är det en länk som medföljer. När personen som fått mailet klickar på länken aktiveras en funktion som flyttar informationen som lades till i den temporära tabellen till User tabellen. Genom att hantera aktiveringen och verifieringen på detta sätt kan man inte logga in förrän man har aktiverat kontot genom mailet. Samtidigt kan man inte bara logga in om man har fått mailet tillskickat sig och inte har skapat ett konto. För att kunna skicka iväg mailet från servern används det en modul som heter Nodemailer.

Användaren inloggad

När användaren har aktiverat sitt konto och loggat in finns möjligheten att ladda upp bilder. Användaren använder sig av ett formulär på sidan där den väljer vilken bild eller fil som ska laddas upp. Där finns två implementationer av hur lagringen kommer att gå till. Genom modulen Multer får man möjligheten att spara bilder och data som är av typen ”multipart/form”. Detta ställs in i Angularjs när den skickar till servern. I Multer finns det ett antal inställningar som man kan göra. Där kan man bland annat säga till att det är bara bilder som ska få lov att laddas upp. Multer hanterar lagringen och det går att visa Multer var den ska lagra filerna. Det mesta den är baserad till är att lagra filerna (bilderna i denna applikationen) lokalt. Däremot har Multer även andra moduler som kan arbeta med till exempel Amazon och lagra dem på deras servrar med. Den andra versionen av Multer är MulterS3 för Amazons S318 lagring.

[17] https://v4-alpha.getbootstrap.com/

[18] https://aws.amazon.com/s3/?sc_channel=PS&sc_campaign=acquisition_SW&sc_publisher=google&sc_medium=ACQ-P%7CPS-GO%7CBrand%7CDesktop%7CSU%7CStorage%7CS3%7CSW%7CEN%7CText&sc_content=s3_e&sc_detail=amazon%20s3&sc_category=Storage&sc_segment=293648113415&sc_matchtype=e&sc_country=SW&s_kwcid=AL!4422!3!293648113415!e!!g!!amazon%20s3&ef_id=Cj0KCQiA68bhBRCKARIsABYUGie920v9iteO_fkwdxG7s3gGxvTbd9Pn87tNLD62sfVAb7qi31NsSvAaAhaXEALw_wcB:G:s

Lagringen brukar för det mesta när man pratar lokalt vara till den datorn som man sitter på när man är på sidan. Det är den första versionen som man kan använda för att spara. I inställningarna i servern för den modulen behöver man ge informationen till ens konto som man har skapat på Amazon så att den ska kunna koppla sig dit. Det gäller även att man har skapat en ”bucket”(hinken). Information om detta hämtas från auth.js filen:

```
/*Amazon information*/
            signatureVersion: 'v4',              //
  secretAccessKey: '',               // insert secret Access  Key
  accessKeyId: '',                   // instert secret Key Id
  region: '',                        // Witch region

  bucket: '',   // the name of the bucket
   acl: '',
   key: '',
  secret: '',
  bucketPath : "",
```

Här sparar man all viktig information som behövs för att skapa en koppling till bland annat Amazon.

I hinken sparas alla bilder. Oavsett om det sparas lokalt eller på andra plattformar, som till exempel Amazon, sparas länken till bilden och laddas upp till databasen. Beroende på var man har databasen måste man ha kopplat rätt information till den modulen som har hand om databashanteringen för att den ska kunna koppla sig till denna databas. Där sparas även det idt som man fick när kontot skapades. Det gör att bilden och personen är sammankopplade. Detta möjliggörs bland annat genom sessionen som har sparat ens identifikation.

Eftersom alla bildernas sökväg, det vill säga var filen ligger i filstrukturen på det system som den är sparad till, är sparad på databasen är det bara att hämta dem från databasen till servern och sedan skicka dem till webbläsaren. I webbläsaren hanteras de först i kontrollerna som har bett om den informationen och sätter in dem i html koden som visar det i webbläsaren. Beroende på vilken användare det är kan det vara olika antal bilder den har laddat upp. Detta spelar ingen roll för genom Angularjs skapar man bara en loop som skriver ut bilderna efter varandra.

```
<div ng-repeat="img in images track by $index">

<div class="card " style="width: 20rem;">

 <img class="card-img-top img-thumbnail" ng-src="{{img.path}}" alt="Card image
cap">

 <div class="card-block">

  <p class="card-text">{{img.name}}.</p>

 </div>

</div> - gelleries.html – line 11 – 18
```

Genom att använda sig av Bootstrap klasser kan vi applicera en mobil design automatiskt. Dessa klasser kan vara till exempel card som man kan se är använd i koden ovanför19.

Bildernas information skjuts in i loopen och med img variabeln som är en del av alla delarna i loopen. Genom detta får varje del sitt eget objekt med information. Och för att visa bilden används det ett Angular ng-src med objektets ”path” information som visar var man kan hitta bilden.

Designen i detta har låg prioriterat eftersom tanken var att det först och främst skulle fungera och det är därför som bilderna är listade under varandra.

Applikationens uppbyggnad i Angurjs är uppbyggd i ett antal olika delar. Först och främst är det index.html filen som är den första biten i bygget. I denna fil läses alla filer som Angularjs behöver för att köra. Även andra filer som hanterar designen med bootstrap och css laddas in.

```
<link rel="stylesheet" href="/css/bootstrap/bootstrap.css">
<script src="/js/jquery.min.js"></script>
<script src="//cdnjs.cloudflare.com/ajax/libs/tether/1.3.1/js/tether.min.js"></script>
<!-- angular link -->
<script src="/js/angular.min.js"></script>
<script src="/upload/ng-file-upload-shim.min.js"></script>
<script src="/upload/ng-file-upload-all.js"></script>
<script src="./libs/angular-touch.min.js"></script>
<script src="./libs/angular-route.min.js"></script>
<script src="./libs/angular-resource.min.js"></script>
<script src="./libs/angular-locale_sv-se.js"></script>

<script src="/js/bootstrap.min.js"></script>
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/font-
awesome/4.7.0/css/font-awesome.min.css">
<!-- Custom styles for this template -->
<link href="css/footer.css" rel="stylesheet">
<link href="css/star.css" rel="stylesheet">


            <!-- Controllers -->
<script src="app.js"></script>
<script src="controller/dashboardCtrl.js"></script>
<script src="controller/newPassCtrl.js"></script>

            <!-- Directives -->
<script src="directives/headerNav.js"></script>
<script src="directives/gelleries.js"></script>
<script src="directives/fileUpload.js"></script>
<script src="directives/footerSticky.js"></script>
<script src="directives/resetPass.js"></script><script
src="directives/starRating.js"></script>
```

Detta gör att webbsidan kan visas.

[19] https://v4-alpha.getbootstrap.com/components/card/

App.js i www katalogen hanterar all routing inom Angularjs samt deklarerar alla moduler som Angularjs behöver.

```
var app = angular.module("myApp", [
  'ngRoute',
  'ngResource',
  'ngTouch'

]);
```

Denna kod definierar alla modulerna som Angularjs kan använda.

```
app.config(["$routeProvider", "$locationProvider", function($routeProvider,
$locationProvider){

            $routeProvider.when("/",{
                        templateUrl: "views/home.html",
                        controller: ""
            })
            $routeProvider.when("/upload",{
                        templateUrl: "views/upload.html",
                        controller: ""
            })
            .when("/gallery",{
                        templateUrl: "views/gallery.html",
                        controller: ""
            })
```

Detta är en del av hur dirigeringen av sidorna ser ut i Angularjs.

Denna hanterar de olika filerna som ska visas beroende på vilken adress man tar sig till.

Kontroller

Utifrån dessa två filerna som bygger stommen i Angularjs uppbyggnad har applikationen även alla de kontroller som hanterar informationerna som hämtas från databasen genom servern. I applikationen finns två kontroller och de är dashboardCtrl.js och newPassCtrl.js.

Direktiv

Utöver de två kontrollerna används det även ett antal direktiv för att hantera informationen både från databasen men även inom applikationen som kan skickas mellan de olika delarna. Man kan välja mellan att använda kontroller eller direktiv. I denna applikation har valet legat oftast på att använda sig av direktiv. Direktiven är uppbyggda genom en js fil där man deklarerar bland annat namnet på taggen som är en koppling till direktivet[20].

```
app.directive('headerNav', [function(){
            //Returning our html template for the menybar.
  return {
    templateUrl: 'directives/headerNav.html',
    controller: ["$scope", "$location", "$http", "$window", function($scope,
$location, $http, $window)
```

[20] https://docs.angularjs.org/guide/directive

Här syns en del av direktivet för huvudmenyn. Här deklareras även vilken html fil(designen) som ska användas. Denna fil fungerar som kontrollerna för här kan man skapa olika funktioner som man kan kalla på i html eller för att få dem att hämta eller skicka information till servern. Servern tar vidare hand om informationen. För att kunna visa dessa direktiv sätter man taggen, som man har döpt i js filen, i andra html filer. Huvudnavigationens taggnamn är headerNav.

I html filen sätter man ett **–** istället för stor bokstav. Det gör att taggen kommer att se ut så här istället: `<header-nav></header-nav>. – index.html – line 46`
Vidare finns det även alla bibliotek som kan behöva användas men även CSS filer för design samt bildmapp för olika bilder inom applikationen.

Genom att applikationen är uppbyggd för att man ska kunna logga in genom att skapa sig ett konto finns det en möjlighet att återställa sitt lösenord om man inte kommer ihåg det. Att återställa lösenordet fungerar på liknande sätt som att registrera sig. Användaren får skriva in sitt email i ett formulär. Denna information skickas till servern och där görs det en koll i databasen om det finns ett sådant email. Om svaret från databasen inte är tom utan att det fanns något skickas denna information till en funktion som skapar en token som är en slags nyckel till det objekt där emailet ligger i. När detta är klart skickas detta till användarens email med en mall som ligger i modules i filen sendpass. I mailet som användaren får ligger en länk med token till en sida där användaren kan byta lösenord.

Analysen

Målet har varit att skapa en applikation samt kunna beskriva vilka val man har gjort för att få den att fungera.

Analysen delas upp i två delar. Den första delen hanterar det ramverket som använts för att kunna utveckla en applikation.

Den andra delen fokuseras mestadels på de olika ramverk som hanterar databasinteraktionen. Finns i alla fall två olika sätt att arbeta med databasinteraktionen som har kommit upp i detta arbete.

Ramverk för att utveckla webbapplikationer

Under denna arbetsprocess har fokuset legat på Angularjs. Det finns ett antal ramverk att arbeta med inom JavaScript men de har man valt att lägga åt sidan och fokusera sig på Angularjs.

Angularjs är utvecklad av Google och som det är beskrivet ovan i texten är den baserad på MVC[21] tänk.

Ramverket är relativt enkla att sätta sig in i och de har bra dokumentation på nätet.

Ramverken för att interagera med databaser

Att arbeta med databasinteraktionen kan göras på många olika sätt. De teknikerna som finns är Vanlig SQL, NoSQL som används för det mesta till Dokumentbaserade databaser som Mongodb samt ORM baserad teknik som är ett annat tänk. Eftersom arbetet har utvecklats med ORM kommer för och nackdelarna att baseras på detta.

Fördelarna med ORM.

De positiva sidorna med att använda sig av ORM baserad teknik är att man får en bra översikt över sin databasmodell samt att den skriver all hantering till och från databasen. I de mindre applikationerna är det ett snabbare och enklare sätt att interagera med databasen för att skicka och ta emot data, samt lägga till och ta bort information. Det går att blanda SQL frågor med ORM tekniken. Tar man Sequlizejs skriver man skriver man så här för att skriva egna frågor till databasen:

```
sequelize.query("UPDATE users SET y = 42 WHERE x = 12").spread((results, metadata)
=> {
```

Nackdelar med ORM

Vad kan man se för nackdelar med att använda sig av ORM tekniken? Genom att enbart se det från en högre nivå av abstraktion så kan man ha svårt att se vad som händer med den kod som implementeras. Eftersom det mestadels sköts av programmet är det svårt att få en kontroll över hur man själv vill att det ska fungera. Man förlitar sig mycket på modeller och kanske tappar fokus över att ha en bra designad databas. Det är enbart vissa parametrar i sökningarna som går att förändra. ORM frågorna är mindre snabba än vanlig SQL samt inte att föredra när man arbetar med mycket stora datamängder.

> *” Compared to traditional techniques of exchange between an object-oriented language and a relational database, ORM often reduces the amount of code that needs to be written.[1]*

[21] https://angular.io/guide/architecture

Detta är också beroende på vilket ramverk inom ORM som används. Märkte att det var en blandad variant av ramverk för ORM inom Javascript och då Nodejs och alla höll varierande standard. Detta är också en tanke att det kan vara svårt att finna ett tillräckligt bra ramverk som gör samma saker som SQL kan göra.[22]

> *Disadvantages of ORM tools generally stem from the high level of abstraction obscuring what is actually happening in the implementation code. Also, heavy reliance on ORM software has been cited as a major factor in producing poorly designed databases.*[23]

[22] https://en.wikipedia.org/wiki/Object-relational_mapping#cite_note-2
[23] https://en.wikipedia.org/wiki/Object-relational_mapping#cite_note-3

Slutsats

Vilka slutsatser har det gått att dra efter detta arbete?

Börjar med att se vad det var för frågeställningar vi hade:

- Vad har man funnit för hjälpmedel som man har använt sig av?
- Hur har arbetssättet fortgått och har hjälpmedlen varit positiva eller negativa.

Hjälpmedel

Genom analysen fick man en bild av positiva och negativa sidor som det hade med sig att använda sig av Angularjs. Att använda sig av Angularjs är enbart positivt om man ser efter tidsvinsten av att kunna det ramverket innan. Eftersom detta var en relativt liten och okomplicerad applikation tror jag inte att man hade fått ut mer av att använda sig av andra ramverk. Samt att Angularjs är supportat till 2020.

Att använda sig av ORM baserade tekniker underlättade eftersom den arbetar på samma sätt som Mongodb och detta är också en teknik som vi lärde oss i skolan. Samtidigt som tekniken lät utvecklaren skriva mindre och ge den lite mindre kontroll. Detta har inte varit någon större förlust under detta arbete eftersom det inte har varit några avancerade datahanteringar.

Arbetssätt

Tidsmässigt har det varit ryckigt av många olika anledningar. Detta har medfört att mesta tiden har försvunnit och nödvändiga prioriteringar av vissa funktioner har behövts göras. Visionen som var tänkt från början blev genom detta förändrad. Däremot har den största funktionen blivit implementerad i applikationen. Men även den har varit tvungen att förändras ett antal gånger under arbetets framfart. Allt som allt tycker jag att arbetet har flutit på i sin takt och med hjälp av flertalet ramverk som man hade en stor förståelse för genom utbildningen kunde arbetet avslutas inom den tid man hade tänkt sig.

Även om man behövde bygga allt från grunden inom server delen var det den som behövde arbetas minst på. Den delen var mestadels aktuell under början när man skulle bygga upp hantering av http förfrågningarna som servern skulle hantera samt funktionerna som de olika modulerna skulle arbeta med.

Ha en struktur över sin teknik. Det har varit blandade ORM tekniker i applikationen. De fungerar liknande men de arbetar efter olika sätt.

Hibernate har bara stöd för Insert och Search till databasen medan Sequlizejs är ett fullutvecklat system med full CRUD funktionalitet. Eftersom man har påbörjat med att implementera Hibernate i systemet innan dess tillkortakommanden syntes är det en blandning av de olika delarna i applikation. Detta har inte påverkat arbetet i säg mer än att man var tvungen att förändra sitt synsätt när man skulle arbeta med Sequlizejs istället.