**Självständigt arbete på grundnivå**

*Independent degree project - first cycle*

Webbutveckling - Mittuniversitetet

*Web development - Mittuniversitetet*

**Addswift**

En webbplattform för alla typer av internetanvändare över hela världen

**Christopher Gauffin**

**MITTUNIVERSITETET**Avdelningen för informationssystem och -teknologi (IST)

**Examinator:** Mattias Dahlgren, [mattias.dahlgren@miun.se](mailto:mattias.dahlgren@miun.se)  
**Handledare:**  Magnus Johansson, [magnus.johansson@sizmek.com](mailto:magnus.johansson@sizmek.com)  
**Författare:** Christopher Gauffin, [christopher.gauffin@gmail.com](mailto:christopher.gauffin@gmail.com)  
**Utbildningsprogram:** Webbutveckling, 120 hp  
**Huvudområde:** Datateknik  
**Termin, år:**VT, 2018

Sammanfattning

Lite del av varje, ej uppslagsverk

Abstract

Sammanfattning på engelska

Terminologi

|  |  |
| --- | --- |
| **Akronymer/Förkortningar** | |
| HTML | Hypertext Markup Language |
| JS | JavaScript |
| CSS | Cascading Style Sheets |
| PHP | Hypertext Preprocessor |
| JSON | JavaScript Object Notation |
| XML | Extensible Markup Language |
| SEO | Search Engine Optimization |
| GDPR | General Data Protection Regulation |
| EU | European Union |
| HTTP(S) | Hyper Text Transfer Protocol (Secure) |
| JWT | JSON Web Token |
| WBS | Work Breakdown Structure |
| NPM | Node Package Manager |
| SPA | Single Page Application |
| SSR | Server Side Rendering |
| AJAX | Asynchronous Javascript and XML |
| API | Application Programming Interface |

# Inledning

Världen globaliseras, individers kontaktnät växer över landsgränser tack vare internet. Men denna globalisering har även lett till att internetanvändare samlar på sig väldigt mycket information. Idag är även internet en stor del av många liv i den äldre generationen. Därför ställer det också högre krav på säkerhet i system, snabbare och bättre teknik samt applikationer.

I stormen av sociala medier så har det nästan blivit för mycket av det goda där man har ett konto för all typ av social media över internet bland annat rekrytering, dating, spel, bilder, videor, bloggar eller forum.

Addswift är ett system som kan hjälpa en användare att bygga en översiktlig profil över konton, data och personliga siter eller bloggar på ett säkert sätt. På detta vis så kan användaren skicka en länk till sina kontakter för att direkt sammanbinda kontakt över alla plattformar.

Kapitel 2, kapitel 3, referenser, tidsplan

## Bakgrund

Det finns många sökmotorer för det internet som är publikt men ingenting för det dolda nätverket deep web (1) som sökmotorer inte kan indexera. Därför behövs en central punkt där användare kan samla och dela med sig av sin information. Det saknas alltså någoting som kan underlätta för internetanvändare att få bättre koll på sina profiler och för att lättare skapa kontaktnät och anslutning till andra personer på ett och samma ställe.

Data lagras överallt, men oftast på ett väldigt osäkert eller inkräktande vis. Äldre tekniker används, lösenord lagras i klartext, osäkra protokoll används. Exempelvis så har 167 miljoner konton har blivit stulna från det stora och välkända Linked In (2) som bara är en av många företag som har blivit kapade på grund av otillräckliga säkerhetsåtgärder. Företag säljer och/eller använder data utan användarens tillåtelse, exempelvis det nyliga Cambridge Analytica fallet där dem använde information av miljontals facebook-användare utan deras vetskap (3).

Varför just detta område, vilka problem ska lösas?

## Övergripande syfte

Projektets röda tråd är skapandet av en enklare prototyp av webbplattformen Addswift vilket är det första steget för att förverkliga idén om ett sammanslutet internet och även för att studera hur ett verkligt projekt skulle kunna genomföras i en professionell miljö från start till slut. Det kommer till en början att genomföras i experiementellt syfte för att undersöka hur denna vision kan uppnås.

Projektet strävar efter kunskap inom affärsplanering alltså hur ett företag fungerar och finansieras, företagets form, marknadsstrategier i form av en affärsplan. En projektplan för att förstå och få erfarenhet i hur ett projekt genomförs med hjälp av diverse tekniker, arbetsätt och metoder. Utveckling av praktiska färdigheter inom design och regelverk för en komplett webbplattform som ska fungera på alla enheter, för alla typer av människor och en användbarhet i alla olika miljöer. En bra förståelse i hur tekniska verktyg, databaser, ramverk, programmeringsspråk, utvecklingsmiljöer och metoder används och fungerar i projektutveckling.

Prototypen har skapats för att göra företaget mer intressant för eventuella investerare i framtiden. När man kan presentera någonting som är mer konkret och som kan visualisera visionen för företaget så blir både produkten och företaget i sig genast mycket mer intressant.

Under projektets gång så har ett antal studier gjorts både för vilka tekniker som är mest moderna och som anses säkrast vilket sedan har används och implementeras som ett ”*Proof of Concept*” för att sedan studera hur dem faktiskt fungerar i praktiken.

## Avgränsningar

Företaget Addswift är ett fiktivt företag och studier har endast genfomförts i ett vetenskapligt syfte där företaget eventuellt i framtiden utvecklas till ett verkligt företag med kommersiellt syfte.

Affärsplanen, *Bilaga B*, är del av en tidigare kurs*, Affärsplaner och kommersialisering* och har presenteras i ett tidigare tillfälle. Den ligger som bas för denna studie vilket är en utveckling av första delmomentet i affärsplanen.

## Mål

Målet med studien är dels att följa den tidsplan som är definerad i projektplanen och att leverera det som anges i produkt och projektkraven för projektplanen vilket är en prototyp av webbplattformen Addswift som ska finnas uppladdad på en webbserver.

Studien vill också upplysa olika teoretiska avsnitt om säkerhet, tillgänglighet, design och utveckling. Målet med detta är att skapa en grund för webbplattformen som är skalbar, använder hållbara tekniker och som anpassas till de riktlinjer som en webbplattform bör innehålla samt följer de regler och lagar för hur data sparas och hur tjänster används.

De frågeställningar studien vill besvara är bland annat:

Hur autentiserar man en användare på ett säkert sätt?  
 Hur implementerar man denna autentisering?

Vilka ramverk fungerar bäst med en plattform som kräver integration med många olika tjänster och som hanterar olika typer av data?  
 Hur implementerar man dessa ramverk?

## Vidd

Tekniskt fokus i denna studie kommer att ligga på NodeJS, MongoDB och Vue. Medans de teoretiska avsnitten kommer att fokusera på säkerhet, ramverk, riktlinjer, lagar och tillgänglighet

## Översikt

## Författarens bidrag

Författaren, Christopher Gauffin har designat och utvecklat prototypen, skapad alla tillhörande bilagor, skrivit projektplan och affärsplan och gjort studier på egen hand. Öppen källkod som ramverk, mallar, plugins har använts vilket författaren inte har utvecklat.

Magnus Johansson har varit tillgänglig för handledning och stöd under projektets gång utan ett aktivt deltagande.

# Teori (background knowledge)

## Teknikstudier

För att börja utveckla webbplattformen Addswift så behövde först en ingående studie göras om de olika ramverken, databaserna och språken för att ta reda på vilka tekniker som är mest relevanta och lämpliga för projektet.

### Fullstack

Först så behöver man förstå sig på vad *fullstack* (4) utveckling innebär. En fullstack utvecklare kan arbeta både med den grafiska presentationen av en applikation på frontend sidan och kan dessutom arbeta på backend sidan med bland annat datalagret och autentisering av användare. När man förstår båda de båda sidorna av applikationen så är det lättare att förstå hur dem kommunicerar och sammarbetar för att få en mer översiktlig bild och förståelse över vilken fil en viss logik hör hemma.

Det finns väldigt många olika metoder för att skapa en fullstack applikation med ett väldigt stort utbud av ramverk, bibliotek och verktyg för att installera, struktuera, kompilera och utveckla applikationen som används för olika syften beroende på projektets ändamål.

### Ramverk

###### Native

Vissa ramverk fungerar bättre för mobila applikationer där har vi exempelvis *Ionic* (5)*,* som bygger på *Cordova* (6) vilket är användbart för att kompilera en *native app* av en webbapplikation.

###### SPA

Andra ramverk som *Angular* (7) och *React* (8) och fungerar bättre för större projekt som främst används i en webbläsare och har ett stort utvecklarstöd och användarbas. *Vue* (9)är en enklare och mer lättviktig variant som strävar efter att inkludera de bästa delarna av både Angular och React.

Applikationer som skapas med dessa ramverk kallas *SPA (Single Page Applikation)* (10) där man endast laddar applikationen en gång i webbläsaren och renderar övriga sidor dynamiskt med AJAX (11) teknologi.

###### SSR

Det finns även ramverk som erbjuder *SSR (Server Side Rendering)* (12)*,* där hela sidan som anropas laddas in direkt istället från servern istället för att klienten hämtar en fil i taget efter det initiala anropet. Exempelvis så används denna metod bland annat i *Laravel* (13) som grundar sig på *Symfony* vilket är skrivet i PHP*,* Ruby on rails, ASP.net för utveckling i Windows miljöer med C#, *Express* (14) vilket bygger på NodeJS/JavaScript eller Django och den enklare varianten Flask i Python.

###### MVC

Oftast i mer väletablerade projekt så används ett *MVC (Model View Controller)* (15) ramverk, där man använder sig av modeller som central komponent som är oberoende av gränsnittet, vyer för att representera information och kontroller för inmatning av data som konverteras till kommandon för vyer och modeller. De ramverk som har nämts ovan grundar sig alla på just denna arkitektur.

Om endast simpel applikation eller statisk hemsida ska skapas däremot så kanske inte det är nödvändigt med ett MVC ramverk. Därför finns det mer lättviktiga bibliotek bland annat jQuery och Bootstrap tillgängligt för att både förenkla skapande av design och layout samt utveckling och manipulering av hemsidans element utan att behöva förbereda en komplex projektstruktur och arbetsflöde.

### Databaser

### Buildsystem

Ett *buildsystem/task runner* inom webbutvecklingär vanligtvis en uppsättning av verktyg/kommandon som kan användas för att förenkla utveckling, effektivisera kompilering och för att optimera, testa och validera en applikation.

###### *Grunt*

Grunt (16) är en äldre task runner som länge var standard för att konfiguera ett automatiserat arbetsflöde och som används fortfarande idag i vissa applikationer. Grunt använder *tasks* eller uppgifter som defineras i filen Gruntfile, detta kan exempelvis vara minfiering av JavaScript eller kompilering av SASS kod. Dessa defineras som ett objekt där man vanligtvis anger ett plugin för uppgiften, ex. jsuglify, vilka filer som används för input, alternativ och destination.

###### Gulp

Gulp (17) kom på senare tid och fungerar och defineras på ett liknande sätt som Grunt. Skillnaden är att Gulp använder sk. pi*pes* istället för ett objekt när man skapar en task, därför att man använder ihopkopplade funktioner ”.pipe()” där man passar in parametrar som exempelvis jsuglify med tillhörande alternativ.

###### Webpack

Med tillväxten av MVC ramverk och den ökande komplexiteten av olika typer av applikationer som använder ES5/ES6 (18) syntax, vilket är JavaScript med utökad funktionalitet som klasser och *lambda* (pilfunktioner). Samt det stora utbudet av moduler och optimerings verktyg, så behövdes ett mer generellt system, Webpack (18).

När Webpack sammanpackar en applikationså bygger den en graf i bakgrunden som kartlägger varje modul som behövs och skapar ett paket, en fil som kallas *bundle*. Detta görs genom att först definiera en *entry* fil, där grafen börjar och även en *output* fil för bundle paketet. Istället för tasks så kedjar man *loaders* för att processera och kompilera koden. Det finns även plugins som *hot-module-reloading*, vilket är en typ av kompilering där den nya koden injeceras direkt i webbläsaren så att ändringar sker utan att behöva ladda om applikationen. (19)

## GDPR

GDPR eller Dataskyddsreformen är en relativt ny lag som godkändes i april 2016 av den Europeiska Unionen (EU) och som inträder 25 maj 2018 vilket har givit företag 2 års tid att förbereda sig till att rätta sig efter lagen. (20)

Lagen kommer att modernisera och ersätta en tidigare lag vilket kallas Data Protection Directive eller Dataskyddsdirektivet som uppfylldes av den svenska Personuppgiftslagen.

GDPR kommer att främst beröra generell behandling av personuppgifter. Det finns också ett förslag om en ny förordning för integritet när det kommer till hur tele- och internetoperatörer får använda data i den elektroniska kommunikationen. (21)

Lagen kommer framförallt göra det svårare för företag att använda sig av oklara och förvirrande små finstilta texter för att lura användaren att samtycka med deras användarvillkor för tillgång till känslig data.

Den strävar också efter att användaren ska ha full kontroll över sin data och ska när som helst kunna radera datan helt och håller från den tjänst där datan har sparats. (20)

Regulationen appliceras inte endast för organisationer som är baserade i EU utan också för dem som är baserade utanför EU men som samlar och hanterar personlig data av individer som lever inom EU.

Personlig data anses enligt den Europeiska Kommisionen vara all typ av data som man kan relatera till en individ, vare sig det är relaterat till hans eller hennes privata, professionella eller offentliga liv. (22)

## Kryptering

I kryptografin så använder man någonting som kallas *salt* för att göra kryptering mer säker. En salt är en splupartad sträng som används som ytterligare input till en funktion som *hashar* datan, exempelvis en funktion som krypterar lösenord från klartext till eller lång sträng av karakärer genom en algorithm. Salt blandas in för att öka komplexiteten och för att skydda mot en *dictionary attack* eller *rainbow table* attack, vilket kortfattat innebär att databasen testas mot de allra vanligaste lösenorden. (23)

## JWT

JSON Web Token, förkortat JWT, är en JSON baserad öppen standard för att skapa *access tokens* vilket är en hash som används för att få en speciell åtkomst till ett system. Exempelvis så kan servern generera en token som är giltig för att en användare ska kunna logga in som admin.

En token är själv behållande, vilket betyder att den kan behålla all den nödvändiga informationen av sig självt. Detta gör att man kan skicka en JWT med en *payload* (vanligtvis användardata) och en signatur.

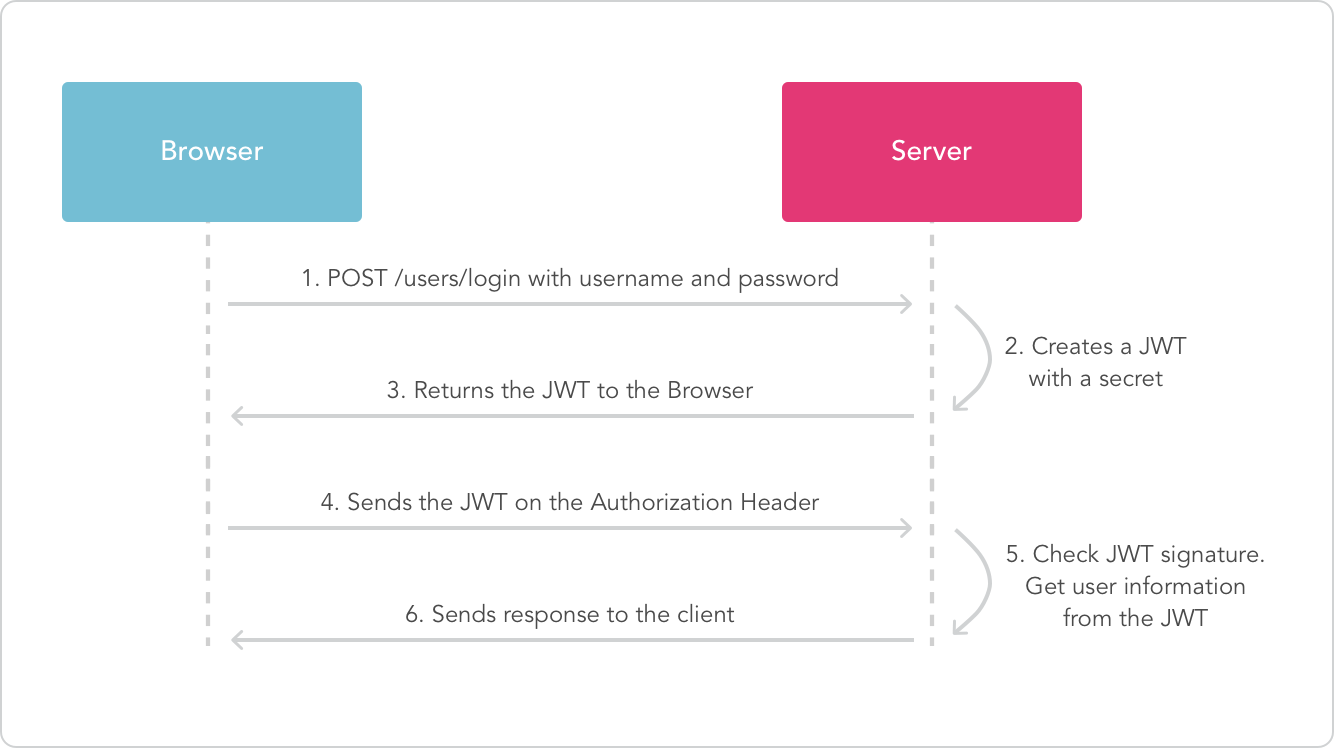
En JWT består av 3 olika delar, en *header* som innehåller typ och hash algoritm, payload, den information man vill skicka med och en signatur som använder headern och payload tillsammans med en hemlig sträng från servern för att generera en hash. (24)

När en användare har lyckats autentisera sig så skickas en JWT till klienten (ex. webbläsaren) från servern. Denna sparas sen antingen i *localStorage* eller som en kaka.

När användaren vill komma åt en skyddad undersida, data eller källa så behöver JWT hashen skickas tillsammans med den HTTP förfrågan som görs till servern vilket görs med en *Authorization* header som använder *Bearer* schemat:

Authorization: Bearer <token>

Följande diagram visar hur en framgångsrik autentisering sker med JWT:



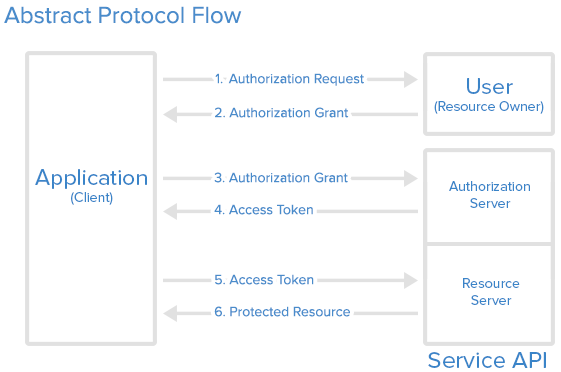
(25)

## Oauth 2.0

Protokollet Oauth 2.0 är en annan industristandard för att autentisering. Fokus ligger på en enkel utveckling på klientsidan men att man samtidigt ska kunna använda protokollet med olika klienter som webb, mobil och desktop applikationer. (26)

Protokollet används för att få tillgång till en användares begränsade rättigheter över en HTTP tjänst, vilket bland annat Facebook, Twitter och Google + använder för deras API tjänster. Det fungerar så att användaren delegeras från klienten till en autentiseringssida för tjänsten (ex. Facebook) där användaren kan acceptera rättigheter för tredjepartsapplikationer (ex. Addswift) att använda kontot för tjänsten. (27)

Nedanför kan vi se ett diagram för flödet hur en användare autentiserar med Oauth:



# Metod

## Affärsplan

Affärsplanen grundar sig istället på hur företaget Addswift är uppbyggt, visionen och de större målen företaget har, hur det finansieras, vilken marknadsstrategi som kommer användas, handlingsplanen och de övriga produkterna som kommer lanseras, en mobilapplikation och en API-tjänst.

Affärsplanen har varit till stor hjälp för att grunda nya idéer och för att skapa ett relevant innehåll. För en mer översiklig bild av företaget Addswift så rekommenderas att den genomskådas, *bilaga B*.

## Projektplan

Projektplanen anger de mer konkreta målen för prototypen av webbplattformen och fokuserar endast på den självständiga studien och inte företaget i sin helhet. Projektplanen fungerade som ett riktmärke för de studier som behövde göras, tidsplaneringen som behövde följas och vilka produkt och projektkrav som behövde uppfyllas. Projektplanen med tillhörande bilagor finns som *Bilaga A* till denna rapport.

## WBS

Den första utmaningen i projektet var att bryta ner det i mindre delmoment och aktiviteter för att få en bättre bild över vad de olika delarna bestod av. Projektet delades upp i tre större kategorier, planering, produktion och avslutning. Detta illustrerades i ett *WBS* (Work Breakdown Structure) schema som skissades i verktyget *Draw.io* (28), bilaga C.

## Gantt

För att se till att projektet alltid utvecklades och så att alla delmoment gjordes i tid så användes ett Gantt schema. Detta gjordes genom verktyget *TeamGantt* (29), där man enkelt kan flytta runt och ändra start och slutdatum för de olika aktiviteterna.

För att veta vilka aktiviteter som behövde planeras i Gantt schemat så var WBS schemat till stor hjälp. Gantt schemat följdes hela tiden under projektets gång för att hinna med deadlines och för att få en överblick över vad som har avklarats och vad som finns framför.

## Trello

Verktyget Trello (30) användes dagligen för att progressivt föra projektet framåt. Trello är perfekt för att få en överblick över var projektet står, vilka problem som behöver lösas och de områden som det berör. Detta gör man genom kort som anger en uppgift vilket flyttas runt beroende på uppgiftens status. Varje aktivitet i Ganttschemat fick var sitt kort i Trello.

För att gruppera kort så kan man använda sig av tabeller, i detta projekt användes 4 olika tabeller. *Todo*, alla de uppgifter som behöver att göras, exempelvis publicering av prototypen, wireframes för design eller ett teoretiskt moment. *Ongoing*, de uppgifter som har påbörjats. *Reporting*, alla de delar som är färdiga men behöver rapporteras. *Done*, uppgifter som är helt avklarade.

I Trello kan man även skriva *bullet points* (listpunkter)*,* viket var till hjälp för att minnas alla olika delmoment, beslut som togs, problem som uppstod och implementationer. För att göra det lättare att gruppera korten så applicerades etiketter som de var relaterade till, projektplan, projektrapport, utformning och prototyp.

## Handledningsmöten

## Utvecklingsmiljö

### Operativssystem

Hela projektet genomfördes med operativsystemen *Windows* 7 och 10 samt *Linux* distributionerna *Kali* och *Arch*. För att få applikationen att fungera på de olika systemen så behövde först NodeJS (31) och MongoDB (32) installeras, antingen genom terminal eller genom grafisk installation. Eftersom webbservern är integrerad i express som körs på NodeJS så behövdes ingen extern webbserver som exempelvis Apache (33) eller Nginx (34).

### Editor

*VSCode* (Visual Studio Code) var den *editor* (kodredigerare) som användes för all kodutveckling. Bland annat så var det möjligt att köra integrerade terminaler direkt i editorn, installera diverse *plugin* (utökad funktionalitet) för *syntax* (kodregler), *snippets* (kodblock) och *autocompletion* (kodtips). Exempelvis *VSCodeVim* för en emulerad Vim (35) navigering och redigering, *Vetur* för Vue syntax eller *Azure Cosmos DB* för direkt uppkoppling mot MongoDB.

### Versionshantering

Med hjälp av Git och github så kunde kod och filer lätt verisionshanteras och överföras mellan de olika enheterna. Med Git så gav det också möjlighet att backa tillbaks till tidigare versioner av projektet vilket gjorde det enkelt att testa olika kodstrukturer, radera filer eller flytta om i mappstrukturen.

### Paketering

*NPM* (Node Package Manager) och det mer funktionella alternativet *Yarn* användes för att installera alla paket som projektet krävde både för utveckling och produktion. Yarn som även använder *cachade* paket (paket som lagras lokalt på datorn) snabbade upp processen vid nya installationer.

# Utformning (designy)

inledning

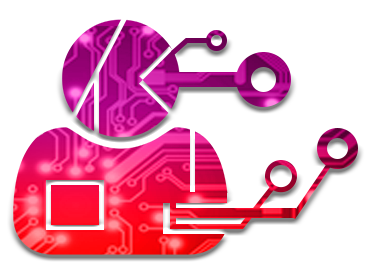
## Logga

Det finns två olika loggor som har skapats, en större standard logga och en annan logga som ikon för exempelvis favicons, profilbild eller appikoner. Båda loggorna har skapats i verktyget *Adobe Photoshop CC* (36)*.*

Den större loggan skapades genom att först ladda ner en font från *Dafont* (37)*.* Med bokstavavstånd och markeringsverktyg så kunde bokstäverna föras samman. Effekter som gradient, avfastning, övertoningsövertäckning och skugga las sen till för att få färg och kontur över texten.



Den mindre loggan gjordes på liknande vis fast istället en font som grund så användes en användar ikon som illustrerar nätverk mellan användare. Genom formverktyget så kunde och markeringsverktyg så kunde ikonen formas om till ett kretskortliknande utséende. Sist applicerades ett extra lager med en bild som maskerades och med en lagereffekt.



## Sitemap

En *sitemap* skapades som första del av designfasen, bilaga D, för att ta reda på vilka undersidor applikationen skulle ha få en idé över hur navigeringen till dem skulle ske. Den är uppdelad i privata undersidor och publika undersidor, där en inloggad användare har tillgång till fler undersidor som exempelvis inställningar.

## Wireframes

Wireframes, enkla skisser på hur applikationens layout skulle se ut skapades när det var tydligare vilka undersidor som behövdes. Dessa befinner sig i mappen bilaga E och skapades med Photoshop.

## ER-diagram

För att designa databasen så gjordes ett ER-diagram. Det skapades två olika diagram, en för projektet alltså prototypen och en annan för vad webbplattformens kompletta databas skulle kunna se ut. Dessa befinner sig under mapp Bilaga F under namnen *Addswift Prototyp ER* och *Addswift ER*.

ER-diagrammet skapades med verktyget *Lucidchart* (38). Diagrammet innehåller de mest relevanta värdena för nuläget och i framtiden är tanken att det ska kunna utvecklas för mer innehåll.

Diagrammet består av entiteter med egenskaper som har relationella sammandband. Varje entitet representerar ett schema i databasen ex. Users, där dess egenskaper är ett objekt med regler, ex. Username som måste vara av typen String. Relationerna mellan entiteterna representeras av linjer där *Information Engineering Style* notationen används för att visa kardinalitet. (39)

## Flödesscheman

Ett flödesschema skapades för att klargöra hur en användare autentiseras för både social media och vanlig inloggning, bilaga G.

Vid normal inloggning så behöver endast kontroll för om användaren existerar och om lösenordet var rätt i utbyte av en JWT.

Vid social inloggning så behöver först rättigheter att godkännas vilket genererar en kod som kan bytas ut mot en *API*-nyckel som kan användas för att hämta informationen och sedan logga in eller skapa en användare. Så här ser Facebooks autentisering ut, den kan skilja sig när det kommer till andra APIer.

## Favicon

En favicon skapades med hjälp av verktyget *favicon-generator* (40) där den mindre loggan användes som input för att generera en favicon.ico fil innehållande bilder för alla olika typer av enheter.

## Storyboard

Ett enklare storyboard, bilaga H ger en generell bild över vilka verktyg, ramverk och miljöer som hela projektet utvecklades med.

# Skapande (techy)

## Vue

Vue var det JavaScript baserade frontend ramverket som användes för att skapa applikationens frontend. Valet gjordes då Vue är enkelt att lära sig och har en väldigt simpel syntax, men kan forftfarande leverera lika bra prestanda som sina motståndare React och Angular.

Det som är bra med Vue är att det är väldigt tydligt hur man skall använda de olika språken tillsammans. En typisk Vue komponent består av 3 olika kodblock. En template där man definerar sin HTML kod precis som vanligt fast med tillgång till kraftfulla attribut som *v-for* vilket kan iterera över ett JS objekt eller andra komponenter som kan innehålla sina egna attribut och logik. Ett skript som innehåller all JS kod, där man bland annat definerar komponentens data, komputerade funktioner, *watches,* metoder eller *hooks*. CSS placeras i en style tagg där man kan välja om koden ska vara *scoped,* endast applicerad för komponenten och om man vill använda en annan typ av syntax ex. stylus, less eller sass.

En komputerad funktion använder en speciell logik för att returnera ett värde och ifall den datan som funktionen använder uppdateras så kommer även det komputerade värdet att uppdateras. En watch fungerar på liknande vis fast där körs istället en funktion varje gång ett värde ändras. En *lifecycle hook* är en metod som kan köras vid ett specifikt tillfälle när en Vue komponent initieras, monteras eller förstörs, Vue har ett bra diagram för detta i deras dokumentation. (41)

Vue använder en modul som kallas *Vue-router* för att genera sk. *routes,* vilket är sökvägar som används för att ladda applikationens olika vyer. Ex. sidan för inställningar har en statisk route ”addswift.com/settings” vilket pekar på komponenten ”settings.vue”. Det finns även dynamiska routes för vyer som inte har en bestämt sökväg som exempelvis en användare, user123, skulle kunna ha sökvägen "/users/user123".

## Vuex

## Stylus

Stylus är en sk. *Dynamic Stylesheet Preprocessor* vilket är ett sätt att använda sig av utökad CSS funktionalitet som sedan kompileras till ren CSS kod. Stylus erbjuder många olika lösningar som gör det mycket mer effektivt att skriva CSS. Bland annat variabler, operatorer, funktioner och mixins. Det går också att nestla klasser och idn under varandra och det är valfritt att använda karakärer som krävs i vanlig CSS; ”{} : ;”. (42)

## Vuetify

För att ge liv till applikationen utséendemässigt så användes ramverket Vuetify (43) vilket baseras på Vue. Vuetify efterliknar Bootstrap men där skillnaden är att Vuetify används i MVC ramverk och där Bootstrap kräver jQuery vilket inte tillhör en MVC mijlö.

Vuetify erbjuder bland annat ett kolumnsystem för att skapa responsiva applikationer, färgteman och typografiska klasser, komponenter som kort, bildgalleri, alerts, menyer och knappar som har ett enhetligt tema.

Dessa komponenter finns direkt tillgängliga i HTML koden precis som vanliga element och noteras med ”v-” ex. istället för en vanlig knapp, *button*, så defineras en Vuetify komponent med *v-button.*

### Responsivitet

För att se till så att applikationen går att använda på alla olika enheter så användes så gjordes applikationen responsiv med hjälp av Vuetify’s kolumnssytem och media queries. Detta är extra tydlig på profilsidan, där layout och menyer byter position och storlek för att passa fönsterstorleken.

Kolumnssytemet baseras på *Flexbox* (44)*,* där strukturen av layouten ser ut som följande *v-container > v-layout > v-flex*. Komponenten *v-flex* tillåter attribut som ändrar storlek beroende på enhet, xs: extra small, sm: small, md: medium, lg: large, xl: extra large, och storlek genom 1-12 där 12 fyller hela bredden på sidan. Så exempelvis om attributen *xs12* och *lg6* anges så kommer behållaren att fylla hela bredden på mobil men bara halva bredden på desktop.

## Nuxt

För att sluta ihop säcken på frontend sidan så användes Nuxt (46) vilket är ett meta ramverk för att skapa komplexa, högpresterande och universala webb applikationer snabbt och smidigt.

### Buildsystem

Vanligtvis så kompileras och utvecklas en Vue applikation med Webpack vilket är utmärkt om man vill ha full kontroll över hela projektet. Men det kan också vara väldigt tidskrävande att sätta upp en komplett och fungerande konfigurationsfil för Webpack.

Nuxt fungerar som ett lager ovanför Webpack som förenklar just denna konfiguration så att utvecklingen fungerar felfritt för en Vue applikation. Detta gör att man får tillgång till kommandon för olika miljöer. Utveckling ”nuxt” vilket startar upp en servermed *hot-reloading*, alltså direkt uppdatering i webbläsaren vid ändring av kod. Kompilering av projektet ”nuxt build” tillsammans med ”nuxt start” för att starta en produktionsserver. Samt ”nuxt-generate” om man vill generera en statisk HTML fil för varje route. Nuxt konfigueras med filen nuxt.config.js medans ett Webpack projekt skulle konfigueras med en webpack.config.js fil.

### Funktionalitet

Förutom att konfigurationen blir betydligt lättare med Nuxt så får man också utökad funktionalitet i Vue. Bland annat så får man tillgång till flera användbara funktioner som exempelvis asyncData som hämtar data innan vyn laddas eller validate som granskar sökvägens parametrar. Man kan också använda egenskaper som *layout* vilket ger en grundlayout för vyn som defineras av en .vue fil placerad i *layout* mappen. Eller *middleware* som definerar om en viss logik ska köras innan vyn visas där logiken placeras i en .js fil i *middleware* mappen vilket användes bland annat för att kontrollera autentisering och admin rättigheter för vyn.

### Routes

Nuxt erbjuder också fördefinerade *routes* genom att placera .vue fil eller en mapp med index.vue i *pages* mappen. På så sätt behöver man inte definera sökvägen med tillhörande komponent i Vue-router utan detta görs automatiskt. Understreck ”\_” används för att notera dynamiska routes, exempelvis ”\_username” används i applikationen för att visa användarens profil där användarnamnet skickas med i ”params” objektet som finns tillgängligt i Nuxt metoderna.

## MongoDB

Databasen som valdes för projektet var MongoDB (47). Anledningen till detta är för plattformen kräver en flexibel och dynamisk databas som kan hantera olika data från alla de API-uppkopplingar som behöver göras för olika typer av tjänster. Om ett konto ska kunna göras för ett ex. ett Facebook konto samt Steam (spel), Soundcloud (musik) så behövs en databas som tillåter att olika data lagras utan att det skapar svårigheter, därför är en dokumentbaserad NoSQL databas som MongoDB perfekt.

### Mongoose

Objekt istället för IDn med populate

Felhantering, Email plugin

Struktur, mappar, subdokument/ObjectId

Typer i databas för skalbarhet, kunna lägga till nya ex. VendorType

För att konvertera ER-diagrammet till tabeller som går att använda i applikationen och för att ansluta till MongoDB så användes pluginet Mongoose (48). Kontruktionen görs genom modeller och scheman. Dessa filer befinner sig i mapparna (*server/models*) respektive (*server/schemas*).

Scheman innehåller alla egenskaper för tabellen samt datatyp och restriktioner (ex. maxlength, required, email, default) med eventuellt felmeddlande ifall egenskapen inte valideras.

En modell är beroende av ett schema. I modellen kan man lägga till övrig logik för tabellen som metoder för att hitta ett ID med en specifk titel eller middleware som skall köras innan en exekvering görs på schemat. Exempelvis så används middleware ”save” varje gång User modellen sparas för att kryptera lösenord med bcrypt.

För att struktuera ett schema så är det även möjligt att definera subscheman i Mongoose, vilket kan användas som meta-data för en egenskap istället för att lagra alla egenskaper i ett och samma schema.

Värt att notera i ER-diagrammet är att vissa relationer pekar på \_id medans andra pekar på entiteten. Detta är för att särskilja scheman som direkt sparar information i subscheman, ex. Account och AccountData, när ett Account raderas försvinner även dess AccountData. Till skillnad från de scheman som är beroende av ett annat schema och pekar därför på \_id, ex. Account behöver en Vendor för att skapas men när Account raderas så behöver fortfarande Vendor (ex. Facebook) finnas kvar.

## Express API

På backend sidan av applikationen är byggt med Express (Node),

## SEO

För att applikationen ska kunna kartläggas av sökmotorer och ge så bra sökresultat som möjligt så har vissa fundamentala åtgärder vidtagits. Detta kallas *Search Engine Optimisation* eller SEO vilket är en slags marknadsstrategi för att växa användarsiffror och uppmärksamhet i resultaten för en sökmotor i en miljö som inte är betald. (46)

### Alt-attribut

Varje bild i applikationen använder Alt-attribut, detta är alltså en sträng som ger en mindre förklaring av bilden. Detta gör att sökmotorn kan på ett smartare sätt förstå innehållet av sidan genom att läsa in attributet.

### Vue-meta

Metadata är data (information) för en större mängd data. I HTML så används ”<meta>” taggen för att definiera metadata och är inte synligt för användaren men kan vara till nytta för maskiner som exempelvis en *crawler* (46) för en sökmotor. (47)

Pluginet Vue-meta har använts för att lägga till relevanta meta-taggar över applikationen. Framförallt så har varje profil (*components/profile/index.vue*) egna meta-taggar vilket läggs till genom head() funktionen där data som följer opengraph (48) protokollet defineras i ett objekt. På profilsidan används titel, typ, url och bild, här skulle även biografi vara relevant om det implementeras på senare tid, vilket skulle resultera i ett komplett sökresultat för varje profil i sökmotorn.

Varje sida använder också en mall för titlen, *titleTemplate*, som är definerat i konfigurationsfilen vilket gör att strängen ”| Addswift” alltid läggs till efteråt, så länge *titleTemplate* inte defineras som *null.* Ex inställningar har titlen ”Settings | Addswfit” medans profilen överskriver mallen och använder bara användarnamn som titel, ex. ”user123”.

### Nuxt SSR

När man använder Nuxt till skillnad från en vanlig Webpack installation så kan man också välja att använda sig av SSR istället för SPA vilket gör att varje undersida laddas på serversidan, vilket gör att sökmotorer kan indexera dem. Exempelvis en profilsida som har sökvägen /users/user123 kan sökmotorn indexera relevant meta information för profilen, medans i SPA skulle sökmotorn bara se startsidan. Detta görs genom att ändra *mode* till ”spa” i konfigurationsfilen.

## Säkerhet

För att autentisera användare så användes JWT teknik genom js pluginet *jsonwebtoken* (46). När en användare loggar in och registrerar sig genom social media så användes istället Oauth teknik, där Facebook inloggning görs genom pluginet *fb* (47). För att validera formulär direkt på klientsidan så användes Vee-validate (52).

### Autentisering

När en användare registrerar och loggar in sig så används ”/register” respektive ”/login” sökvägarna under (*routes/auth/index.js*). När en användare registrerar sig så kontrolleras först att det inte finns någon användare med samma användarnamn i databasen, sen sparas användaren där lösenordet krypteras genom pluginet *bcrypt* (48). Vid inloggning så dekrypteras det lagrade lösenordet för det angivna användarnamnet med det angivna lösenordet och om det matchar så går det vidare till JWT signering.

Implementering av JWT görs genom en middleware fil (*server/middleware/jwt.js*) där 2 express middleware funktioner har definerats, den ena *verifyToken* kontrollerar att en token är giltig genom att bryta ner headern och hitta hashen som verifieras genom jwt.verify för att ge rättigheter tillbaks till klienten. Den andra funktionen *signToken* används för att signera en ny token med jwt.sign där den inloggade användarens ID skickas som payload som sedan skickas tillbaks till klienten.

På klient sidan så sparas och hämtas JWT hashen som en kaka genom Vuex modulen (*store/auth.js*) där (*services/AuthenticationService.js*) kommunicerar med API tjänsten och (*services/TokenService.js*) lagrar kakan och konfiguerar headern.

### Facebook SDK

### Vee-validate

Vee-validate är ett Vue plugin som används för att ge direkt feedback till användaren när dem fyller i ett formulär. På så vis så undviker man onödiga förfrågningar till backend servern och användaren behöver heller inte klicka ”skicka” varje gång för att se vad dem har gjort fel. Pluginet har använts för login (*pages/login/index.vue*) och registrering (*pages/register/index.vue*).

Valideringen defineras med attribut på de fält som ska valideras. ” v-validate” bestämmer vilka typer som ska valideras, exempelvis om det krävs en email med max 30 karaktärer så skulle attributet hålla strängen ”required|email|max:30”. ”:counter” visar hur många karaktärer som har använts, ”:error-messages” håller de felmeddelanden som ska visas och ”data-vv-name” definerar namnet på fältet.

Det som är smidigt med Vee-validate är att den genererar felmedellanden automatiskt genom ”v-validate” strängen. Men det går också att definiera egena felmedellanden i ett objekt, ”dictionary”, där ”data-vv-name” används för att specifiera vilket fält som ska ha speciella regler.

## Kontaktsystem

Feed

IDn vs Populering

Hämta endast nödvändig data med mongoose, ex populate followers (id, username picture)

## Underhåll

Validering, kommentering

## Tillgänlighet

Testning webbläsare hastighet, Färgkoder, textstorlek, riktlinjer

## Publicering

Heroku

# Resultat

## Handledningsmöten

### Första mötet

### Andra mötet

### Tredje mötet

## Profil

Inställningar, publik/privat

Populering av followers för att arbeta med IDn och för att inte ladda in onödig data

Komponenter för att dela upp profil content

Profilen är uppdelad i Vue komponenter för tydligare struktur (*components/profile*). Här är även de olika undersidorna för profilen, accounts, sites och followers upp delat i egna mappar.

Varje användare får sin egna profil när dem registrerats sig. En användare som registrerat genom socialt media får även sitt första konto skapat automatiskt. Funktionalitet för att skapa nya konton och siter har inte implementeras än.

Inställningar har ingen effekt på kontot då detta endast illustrerar hur en användare skulle kunna ställa in sina egna preferenser.

## Admin

Egen route för bättre säkerhet

Middleware för alla routes för att se till att bara admins kan använda dem

Filtrering med computed istället för separata listor

Separat service

Bannlista

En användare som har administrationsrättigheter har även tillgång till en admin vy för att bannlysa användare som inte också har Admin rollen. Detta görs genom att först generera en lista med alla användare i vyn (*pages/admin/users.vue*) . Listan delas sen upp i computed egenskaper; banned, admins och normal. API logiken för dessa

## Tidsplanering

Skalbarhet, mycket saknas med det finns planering

## GDPR

\* Visa att det finns planer för implementering och tanke bakom

\* Delete knapp ge användare full kontroll

Privacy policy + terms of service, registrering

## Web 2.0

SPA Vuejs, Vuex store

Dynamiska anrop för all data, paginering i framtiden

Responsivitet

ES6

Simpel design, ikoner

Framtidsplaner, blockchain, Web 3.0 etc.

# Diskussion

## Förbättringar

paginering

## Sökmotor

Det som saknades, men finns i wireframe, Google fast för konton

## Blockchain

## GDPR

Gdpr, ip med användarinfo?

Stepper i social registrering