YRKESHÖGSKOLAN
NOVIA

Utveckling av en webbapplikation med PHP, MVC och OOP

Alexander Tallqvist

Examensarbete för tradenom (YH)-examen

Utbildningsprogrammet i Informationsbehandling

Raseborg 2018

EXAMENSARBETE

Författare: Alexander Tallqvist

Utbildning och ort: Informationsbehandling, Raseborg

Handledare: Kim Roos

Titel: Utveckling av en webbapplikation med PHP och MVC

Datum 20.01.2019 Sidantal 40 Bilagor 0

Abstrakt

I detta arbete diskuteras utvecklingen av en webbapplikation, konstruerad med serverskriptspråket PHP. Arbetet tar också upp applikationens utvecklingsprocess, och diskuterar bl.a. planeringen av applikationen samt dess versionshantering. Diverse verktyg och teknologier som användes vid utvecklandet av applikationen, tangeras också.

Applikationens back-end konstruerades med hjälp av arkitekturmönstret MVC (engelskans Model-View-Controller), samt programmeringsmetoden OOP (engelskans Object-oriented programming).

Arbetet berättar också om CSS förlängningsspråket SASS, samt PHP mallmotoren TWIG. Webbapplikationens vyer och stilfiler kom att utnyttja både SMACSS (engelskans Scalable and Modular Architecture for CSS) och BEM (engelskans Block-Element-Modifier) arkitekturmönstren.

Till den slutliga applikationen implementerades CRUD funktionalitet (engelskans Create, Read, Update, Delete), med möjligheten för besökarna att registrera sig som användare till applikationen.

Språk: Svenska Nyckelord: PHP, MVC, OOP, BEM, SMACSS

OPINNÄYTETYÖ

Tekijä: Alexander Tallqvist
Koulutus ja paikkakunta: Tietojenkäsittely, Raasepori
Ohjaaja(t): Kim Roos

Nimike: Web-sovelluksen kehittäminen PHP:n, OOP:n ja MVC-arkkitehtuurin avulla

---

Päivämäärä 20.01.2019 Sivumäärä 40 Liitteet 0

Tiivistelmä

Tämä opinnäytetyö käsittelee PHP-ohjelmointikielellä toteutetun web-sovelluksen kehitysprosessia. Työ kertoo muun muassa sovelluksen rakenteesta, suunnittelusta, sekä tämän versionhallinnasta. Työ käsittelee sovelluksen kehittämisessä käytettyjä työkaluja ja teknologioita.

Sovelluksen back-end rakennettiin käyttämällä MVC- arkkitehtuurimallilla (englannin Model-View-Controller), sekä OOP-ohjelmointimenetelmää (englannin Object-oriented programming).

Työ kertoo myös CSS laajennuskielestä SASS, sekä PHP:n mallimoottorista Twig. Sovelluksen näkymissä ja tyylitiedostoissa käytettiin sekä SMACSS- (englannin Scalable and Modular Architecture for CSS) että BEM-arkkitehtuurimalleja (englannin Block-Element-Modifier).

Lopullinen sovellus sisältää CRUD-toiminnot (englannin Create, Read, Update, Delete), sekä antaa sovelluksen vierailijoille mahdollisuuden rekisteröityä tämän käyttäjiksi.

Kieli: Ruotsi Avainsanat: PHP, MVC, OOP, BEM, SMACSS

BACHELOR'S THESIS

Author: Alexander Tallqvist
Degree Programme: Business Information Technology, Raseborg
Supervisor(s): Kim Roos

Title: Development of a Web Application with PHP, MVC and OOP

Date 20.01.2019 Number of pages 40 Appendices 0

Abstract

This thesis discusses the development of a web application, constructed in the server scripting language PHP. The work also addresses the application's development process and discusses among other things the planning of the application, as well as its version control. Various tools and technologies used in the development of the application are also addressed.

The application's back-end was constructed using the MVC architecture pattern (Model-View-Controller), as well as the OOP (Object-oriented programming) programming method.

The work also talks about the CSS extension language SASS, as well as the PHP template engine TWIG. The web application's views and style files came to utilize both the SMACSS (Scalable and Modular Architecture for CSS) and BEM (Block-Element-Modifier) architectural patterns.

The final application implements CRUD functionality (Create, Read, Update, Delete), with the possibility for the visitors to register as a user for the application.

Language: Swedish Key words: PHP, MVC, OOP, BEM, SMACSS

Innehållsförteckning

1 Inledning och syfte

Efter att jag inlett mina studier på Novia Raseborg, så gick det inte länge innan jag hittade ett stort intresse för programmering, och mera exakt, webbutveckling. Innan jag hunnit utföra ens en tredjedel av mina studier till slut, så hade jag redan skaffat mig ett arbete inom branschen, där jag arbetade med innehållshanteringssystemet Drupal, ett CMS (engelskans content management system) skrivet i serverskriptspråket PHP.

Vid ett senare skede började jag reflektera över mitt eget kunnande som webbutvecklare, och ansåg att jag kanske hade fallit i en fallgrop som i dagens läge verkar vara väldigt vanlig för nya webbutvecklare; Jag hade lärt mig använda en spikpistol, innan jag hade en grundläggande förståelse över hur man använder sig av en vanlig hammare och spik.

I dagens läge är webben fullproppad av olika verktyg, CMS och andra teknologier, vars grundläggande uppgift brukar vara att underlätta med själva utvecklingsprocessen av webbapplikationer. Själv anser jag detta vara en bra sak, eftersom att jag kan se en nytta med att inte behöva återuppfinna hjulet för varje gång man vill lägga upp en ny webbsida eller utveckla en webbapplikation.

Dock befann jag mig själv i en situation där jag kunde utveckla relativt komplicerade webbaserade sidor och applikationer i Drupal, och viste t.ex. hur man går till väga för att utveckla nya moduler och teman i systemet. Jag hade alltså en förståelse för *hur* saker och ting skulle implementeras med verktyget som jag valt mig att lära, men saknade ofta svaret på *varför* allting implementerades så som det gjordes.

Arbetets grundläggande syfte är därför att förbättra mina kunskaper inom webbutveckling. De mest väsentliga målen är att förbättra min förståelse för PHP och MVC arkitekturmönstret samt att skapa en förståelse, för hur de olika teknologierna och verktygen som bygger upp en webbapplikation hänger ihop. Arbetet kommer också att tangera olika front-end teknologier, som t.ex. förlängningsspråket SASS, PHP mallmotorn TWIG, samt front-end arkitekturmönstren SMACSS och BEM.

Planen är att använda de olika teknologierna och arkitekturmönstren för att utveckla en relativt enkel CRUD applikation, med användarregistrering.

2 Verktyg och teknologier

I detta kapitel diskuteras programmeringsspråken samt teknologierna som använts till utvecklingen av applikationen. Jag kommer att tangera verktygen på en ganska grundläggande nivå, och försöker samtidigt lyfta fram orsaken till varför verktygen eller teknologin blivit valda just för det här arbetet.

2.1 PHP

PHP (PHP: Hypertext Preprocessor) är ett av många skriptspråk som kan köras på serversidan av webbaserade applikationer och webbsidor. PHP koden tolkas på webbservern för varje gång som en webbsida besöks eller exekveras. Det genererade resultatet skickas oftast tillbaka till webbläsaren som HTML, men format som vanlig text eller JSON (engelskans JavaScript Object Notation) accepteras också. (Welling & Thomson 2017, 2)

PHP språket skapades ursprungligen år 1994 av Rasmus Lerdorf, och kallades på den tiden för ”Personal Home Page”. Därefter har PHP gått igenom flera stora utvecklingsskeden och omskrivningar för att vidareutveckla språket och dess funktioner. För tillfället är huvudversionen för PHP version 7. PHP projektet utvecklas med öppen källkod, vilket betyder att vem som helst har möjligheten till att modifiera och vidareutveckla egna version av språket, utan någon kostnad. (Welling & Thomson 2017, 2–3)

I dagens läge är PHP ett av de mest använda serverskriptspråken i världen. Orsaken till populariteten ligger i språkets starka sidor; PHP har en god prestanda, det är mycket skalbart samt relativt enkelt att lära sig. Enligt Google kördes upp till 75% av alla världens webbsidor och webb baserade applikationer med PHP år 2013, och hädanefter har talet vuxit till 82% år 2016. (Welling & Thomson 2017, 2–5)

2.1.1 PHP Standards Recommendations

Applikationen kommer att sträva efter att ta hänsyn till PSR (PHP Standards Recommendations). Standarderna är utvecklade av gruppen ”PHP-FIG” eller ” Framework Interoperability Group”, som finns till för att samla gensamheterna i olika PHP ramverk, och på grund av dem bygga upp standarder och rekommendationer, som alla kan följa vid utvecklingsskedet av sina applikationer och ramverk. Till projekt som understöder verksamheten hör bl.a. Drupal, Symphony, Joomla och Magento. (PHP-FIG 2018)

2.1.2 Composer

Applikationen använder sig av Composer som pakethanterare. Composer är ett terminal baserat verktyg som används till att installera tilläggsbibliotek för PHP baserade webbapplikationer. För att Composer ska kunna köras krävs en PHP version på minst 5.3.2. (Composer u.å.)

2.2 Apache

Webbapplikationen kommer att använda Apache som sin webbserver. Webbserverns preliminära uppgift är att motta http förfrågningar, och där efter besvara förfrågningen med de material som tillfrågats. Apache användes år 2011 av nästan 50% av alla världens webbsidor, detta tack vare sin förmåga att ihopkopplas med andra applikationer, sin robusthet, samt möjligheten att köra programmet på olika operativsystem. Apache utvecklas dessutom också med öppen källkod, av Apache Software Foundation. (Ljubuncic 2011, 15)

2.3 MySQL

Applikationen använder sig av MySQL som sin databas. En databas grundläggande funktion är att lagra och strukturera information på ett sådant sätt, att det snabbt går att sökas upp och hämtas tillbaka till användaren (Nixon 2013, 171). MySQL är världens populäraste databasapplikation, och används oftast tillsammans med PHP. Förutom programvaran för själva datalagringen ingår också diverse applikationer som tillåter användaren att administrera databasen. (Ullman 2018, 113)

2.3.1 Databasstruktur

En MySQL databas är uppbyggd av flera olika tabeller, som i sin tur innehåller rader med data. En rad med data är oftast konstruerad av flera olika kolumner, som i sin tur innehåller fält med information. Genom att använda oss av namnen för tabellerna och kolumnerna kan vi med hjälp av t.ex. SQL göra förfrågningar från databasen. (Nixon 2014, 171–172). Figur 1 presenterar ett exempel på en MySQL tabell, som i detta fall innehåller information angående anställd personal på ett företag. Tabellen innehåller total sex rader med data, och varje anställd person har utsetts med ett id, ett för- och efternamn, en jobb kod, samt en lön och telefonnummer.

Employees Table

IdNum	LName	FName	JobCode	Salary	Phone
1876	CHIN	JACK	TA1	42400	212/588-5634
1114	GREENWALD	JANICE	ME3	38000	212/588-1092
1556	PENNINGTON	MICHAEL	ME1	29860	718/383-5681
1354	PARKER	MARY	FA3	65800	914/455-2337
1130	WOOD	DEBORAH	PT2	36514	212/587-0013

Figur 1. Exempel på en MySQL tabell.

2.3.2 SQL

SQL står för ”Structured Query Language” i MySQL, och används preliminärt för att göra förfrågningar från t.ex. en webbapplikations eller en webbsidas server till en databas. Förfrågningarna kan innehålla satser för att skapa, modifiera, hämta och ta bort data från databasen. SQL liknar i synnerhet det engelska språket, och ett SQL kommando kan exempelvis se ut såhär: ”SELECT FName FROM Employees WHERE IdNum = 1114;”. I det här fallet skulle förfrågningen försöka hämta värdet från FName *kolumnen* på en sådan *rad*, som har värdet 1114 under IdNum *kolumnen*. (Nixon 2013, 171–172)

2.4 HTML

HTML står för HyperText Markup Language, och är internets preliminära märkspråk (engelskans markup language). HTML skapades i tiderna för att enklare kunna beskriva vetenskapliga texter, men har hädanefter adapterats för att beskriva olika sorts dokument och applikationer. I dagens läge är standarden för webben HTML5, vars slutliga version publicerades år 2014. (W3C 2017). Kodexempel 1 innehåller ett enkelt HTML-dokument med olika elementtyper och strukturmärken.

Kodexempel 1. Exempel på ett enkelt HTML dokument.

```
<!DOCTYPE html>
<html>
  <head>
    <title>Sample page</title>
  </head>
  <body>
    <h1>Sample page</h1>
    <p>This is a <a href="demo.html">simple</a> sample.</p>
    <!-- this is a comment -->
  </body>
</html>
```

2.5 Twig

Twig kommer att användas som applikationens mallmotor (engelskans template engine). Twig publicerades år 2009 av Fabien Potencier, och marknadsför sig själv som en snabb, säker och flexibel mallmotor. Malmotorns preliminära uppgift är att tillsammans med HTML separera presentationen från programlogiken, samt att göra det lättare att hantera och skriva ut data i applikationens eller webbsidans vyer. (Twig u.å.). I kodexempel 2 kan vi se skillnaden på utskrivningen av variabler i PHP och Twig, och ett exempel på en for-each lopp i implementerad med Twig.

Kodexempel 2. Exempel på skillnader mellan PHP och Twig.

```php
<?php echo $var ?>
<?php echo htmlspecialchars($var, ENT_QUOTES, 'UTF-8') ?>
```

```twig
{{ var }}
{{ var|escape }}
{{ var|e }}          {# shortcut to escape a variable #}
```

```twig
{% for user in users %}
    * {{ user.name }}
{% else %}
    No users have been found.
{% endfor %}
```

2.6 CSS

CSS står för ”Cascading Style Sheets”, och tillåter oss att kontrollera utseendet av webbapplikationens olika element, som t.ex. färger, fonter och layouts. CSS tillåter oss också att ändra på applikationens utseende, beroende på apparatens skärmstorleks som använder sig av applikationen. (W3C u.å.)

2.6.1 SASS

SASS (Syntactically Awesome Style Sheets) är ett förlängningsspråk (engelskans extension language) för CSS. SASS tillåter oss att stycka upp vår CSS kod enligt de komponenter som webbapplikationen eller webbsidan representeras av. SASS erbjuder oss även möjligheten att använda oss av funktioner som inte är möjliga med vanlig CSS, som t.ex. variabel, operatorer, funktioner (mixins), samt importering av CSS kod från en fil till en annan. (SASS u.å.). Kodexempel 3 innehåller en SASS mixin som implementerar en CSS deklaration ”border-radius” med olika leverantörprefix (engelskans vendor prefix).

Kodexempel 3. Exempel på en SASS funktion (mixin).

```
@mixin border-radius($radius) {
  -webkit-border-radius: $radius;
     -moz-border-radius: $radius;
      -ms-border-radius: $radius;
          border-radius: $radius;
}

.box { @include border-radius(10px); }
```

2.6.2 Arkitekturmönster för applikationens CSS

För att enklare kunna organisera och strukturera webbapplikationens CSS, har jag valt att använda mig av två olika front-end arkitekturmönster för att underlätta processen. Den grundläggande tanken bakom användningen av front-end arkitekturmönster är inte bara att underlätta med den initiala utvecklingsprocessen, utan också att göra det enklare att bearbeta och möjligtvis vidareutveckla applikationen vid ett senare skede. För denna applikation har jag valt att kombinera arkitekturmönstren SMACCS och BEM.

2.6.3 SMACSS, Scalable and Modular Architecture for CSS

SMACSS är ett arkitekturmönster eller en generell guide som i ett nötskal strävar efter att kategorisera och inkapsla applikationens diverse CSS deklarationer. Webbsidor och applikationer som använder sig SMACSS brukar generellt sett dela upp sina CSS deklarationer i en (1) av fyra (4) olika huvudgrupper. Enligt SMACSS modellen är dessa huvudgrupper ”Base rules”, ”Layout rules”, ”Modules” och ”State rules”. (SMACSS u.å.)

2.6.4 BEM, Block Element Modifier

BEM eller “Block Element Modifier” beskriver sig själv som en front-end metodologi vars mål är att underlätta skapningen av återanvändbara komponenter vid utvecklingsskedet av webbapplikationer. Applikationer som använder sig av BEM strävar efter att uppdela sina CSS deklarationer i s.k. ”*Block*”, som sedan fylls på med diverse ”*Element*”. (BEM u.å.). Figur 2 presenterar ett exempel på BEM metodologin, där de olika blocken är inramade med rött, och elementen är inramade med blått.

Figur 2. Exempel på BEM metoden – Block och Element.

2.7 JavaScript

JavaScript är ett skriptspråk som preliminärt används på klientsidan av webbapplikationer och webbplatser. JavaScript tillåter oss att skapa dynamiska applikationer där t.ex. webbplatsens innehåll han uppdateras utan att användaren behöver ladda webbplatsen på nytt. JavaScript tillåter oss även att köra kod som en respons för något som användaren gör på webbplatsen, t.ex. då hen klickar på en meny eller fyller i ett formulär. (MDN 2018)

2.7.1 ECMAScript 2015

Standarden för JavaScript kallas för ECMAScript. År 2015 publicerades den sjätte (6) huvudversionen av ECMAScript, som officiellt kallas för ECMAScript 2015 eller ES6. Den nya standarden erbjuder ny funktionalitet samt syntaktiskt socker (engelskans syntactic sugar) för att underlätta programutvecklingsprocessen med språket. Till den nya funktionaliteten hör bl.a. moduler, klassdeklarationer samt löften (promises) för asynkronisk programmering. (ECMA u.å.)

2.7.2 Node JS

Node JS är en plattform som möjliggör körningen av JavaScript på serversidan. Node är konstruerad på Googles open-source V8 JavaScript-motor, vilket betyder att då Google implementerar ny JavaScript funktionalitet på klientsidan, så överförs de nya egenskaperna också till Node. Plattformen är utvecklad till att vara bra på att sköta asynkron trafik. Detta betyder att Node är ett utmärkt val vid situationer där systemet förväntas samtidigt kunna hantera t.ex. flera uppkopplingar till en databas, eller läsning och skrivning till ett filsystem. (Codecademy u.å.)

Vid utvecklingen av denna applikation kommer Node JS preliminärt att används tillsammans med dess pakthanterare, NPM, för att installera och köra s.k. JavaScript ”*Task Runners*”.

2.7.3 NPM - Node Package Manager

NPM beskriver sig själv som en pakethanterare för JavaScript baserade applikationer, samt världens största mjukvaruregister. NPM används preliminärt för att ladda ner och till att dela olika kodpaket vid utvecklingsprocessen av webbaserade applikationer. (NPM u.å.)

Vid utvecklingen av denna applikation kommer NPM preliminärt att används för installationen av s.k. JavaScript ”*Task Runners*”.

2.7.4 Gulp.js

Gulp.js är en JavaScript ”*Task Runner*” som används till att automatisera vissa front-end baserade arbetsmoment, som utan ett verktyg som Gulp skulle kräva en hel mängd repetition. Gulp kan bl.a. används till att automatiskt kompilera SASS till vanliga CSS filer, eller till att t.ex. optimera bilder och JavaScript. Gulp används med att man skriver instruktioner i en konfigurationsfil som namnges gulpfile.js. (CSS-tricks 2015)

2.8 Versionshanteringssystem med Git

Versionshanteringssystem är programvara vars preliminära uppgift är att övervaka och hantera förändringar i ett mjukvaruprojekt över en längre tid. Versionshanteringssystem kommer att hålla koll på alla ändringar som någonsin gjorts i projektet, och tillåter dess utvecklare att gå tillbaka i tiden för att t.ex. jämföra äldre versioner av projektet med den nyaste versionen. Versionshanteringssystem tillåter oss alltså inte bara att göra säkerhetskopior på våra mjukvaruprojekt, utan ger oss även möjligheten att ladda upp en

äldre version av projektet, om t.ex. någonting går sönder då projektet är i sitt utvecklingsskede. (Atlassian u.å.a)

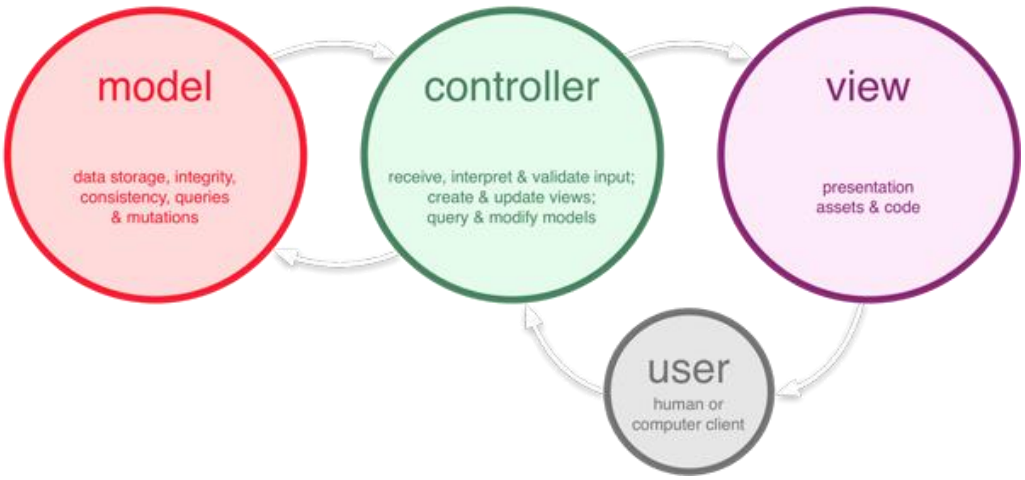
Git är ett versionshanteringssystem som publicerats av Linus Torvalds år 2005. I dagens läge är Git världens mest populära versionshanteringssystem, och används både i kommersiella projekt samt projekt som drivs med öppen källkod. Gits popularitet härstammar från dess starka sidor; Gits prestanda, flexibilitet och säkerhet är tre karaktärsdrag som oftast tas upp då Git jämförs med andra alternativ. Git kan användas via grafiska användargränssnitt och kommandotolken. (Atlassian u.å.b)

3 Arkitektur och programmeringsmetod

I detta kapitel behandlas webbapplikationsens arkitekturmönster MVC, programmerings-metoden OOP (engelskans Object-oriented programming), samt nyttan som förekommer då de används vid utvecklingen av applikationer och webbsidor.

3.1 MVC

MVC eller Model-View-Controller är ett populärt arkitekturmönster som används vid programutveckling, och som oftast sammankopplas med ett stark fokus på objektorienterad programmering. Komponenterna som används vid utvecklingen av applikationen är models, views och controllers, och tillsammans har det som uppgift att bygga upp helheten som är applikationen. (Pitt 2012, 1). Figur 3 presenterar exekveringslogiken bakom MVC arkitekturmönster.



Figur 3. Model-View-Controller arkitekturmönster.

3.1.1 Model

Modellen kan anses vara komponenten som utför allt arbete i MVC trion. Arbetet kan anses vara allting mellan en uppkoppling till en databas, till hantering av tredje parters tjänst, eller för tillkörningen av annan logik som applikationen kan behöva. (Pitt 2012, 1)

3.1.2 View

En view finns till för att bygga upp ett element eller en komponent i användargränssnittet som applikationens användare ser. Alla komponenter som användaren ser och hanterar befinner sig i en view, och oftast är en sida i en webbapplikation konstruerad av flera olika views. (Pitt 2012, 1)

3.1.3 Controller

Controllern är komponenten som ihopkopplar modellen och viewn. Controllerns uppgift är alltså att isolera logiken från modellerna och komponenterna (views) i användargränssnittet från varandra, och fungerar samtidigt som en brygga mellan de två olika delarna. Controllern är alltid den första komponenten som tillkallas i applikationen, eftersom att alla förfrågningar först körs till controllern, som sedan avgör vilka modeller och views som behövs för att uppfylla och svara på förfrågningen. (Pitt 2012, 1)

3.1.4 Routing

Vid exekveringen av en webbsida eller applikation som är utvecklat enligt MVC mönstret, kommer den alltid att behöva kalla på en controller, modell och view. Koden och elementen som bör laddas för att uppfyll användarens förfrågning avgörs via URL:n som hen besöker, i en process som kallas routing. Processen går ut på att koppla ihop olika URL-adresser till diverse controllers och deras metoder. Exempelvis skull adressen ”www.webiste.com/storys/all” kunna kopplas till en ”Story” controller, som är utrustad med en ”All” metod. Metoden skulle sedan ansvara över att koppla sig till de nödvändiga modellerna och vyerna för att uppfylla användarens förfrågning. (Pitt 2012, 2&67)

3.1.5 Varför MVC?

Webbapplikationer och webbsidor byggs oftast upp av tre olika aspekter; Design, äffärslogik samt en kontinuerlig process som exempelvis strävar efter att förbättra, uppdatera eller fixa buggar i projektet. I en ostrukturerad applikation har dessa tre områden en tendens till att

smälta samman till en osammanhängande röra. I både stora och små företag måste oftast flera utvecklare engagera sig för att kontrollera och dubbelkolla att ändringar i en del av programmet inte omedelbart bryter ner andra delar av programmet. Det här är det centrala problemet som MVC arkitekturmönstret vill ta itu med. Mönstret definierar strikta behållande mellan programmets kod och funktioner, och när exempelvis ändringar gällande databaskod görs i en logiskt isolerad modell, så kan man nästan garantera att funktionaliteten i näraliggande controllers och views inte kommer att gå sönder. (Pitt 2012, 3)

3.2 Objektorienterad programmering

Objektorienterad programmering är en programmeringsmetod där man strävar efter att bryta ner programvaran i skilda moduler eller *klasser*. Separeringen av koden i klasser möjliggör för en utvecklingsprocess där koden som skrivs är självständig eller inkapslad, vilket i sin tur gör dess underhållning och uppdatering betydlig enklare, samt underlättar med felsökning. (Ullman 2013, 120)

3.2.1 Nyttan med OOP

För att anse nyttan med objektorienterad programmering, kan det vara värt att först ta en titt på procedurell programmering (engelskans ”Procdural programming”). Programvara vars kod utvecklas enligt den procedurella stilen exekveras linjärt. Koden löper alltså uppifrån neråt, från topp till tå, och kan anses vara en serie steg som utförs efter varandra. Kod som skrivs enligt denna stil kan fungera bra om slutprodukten behåller sig till några hundra rader kod, med vid situationer där kodbasen växer blir den oftast svår att både hantera och felsöka. (Clark 2013, 8)

Principen för objektorienterad programmering har i tiderna utvecklats för att göra själva programmeringsprocessen enklare. Huvudprincipen är att uppdela kod i små diskreta bitar, som kan innehålla både beteenden och data tillsammans. Andra aspekter där nyttan med objektorienterad programmering stiger fram, speciellt när man tar hänsyn till utvecklingstiden och underhållningskostnaderna, kan betraktas enligt följande:

- **Återanvändbarhet**: Kod som skrivs enligt OOP-principen indelas i skilda klasser. Klasserna innehåller egenskaper och metoder som kan vara självständiga, eller utvecklade för att utföra ett arbete med andra klasser. En klass utvecklas oftast för att lösa ett specifikt problem, vilket betyder att då andra utvecklare stöter på samma

problem, så kan de oftast integrera andras lösningar in i sina egna projekt, utan att oroa sig över att den befintliga koden skulle sluta fungera.

- **Utvidgningsbarhet**: Vid situationer där mjukvaruprojekt vidareutvecklas är det oftast relativt enkelt att tillägga ny funktionalitet, om kodbasen är skriven enligt OOP-principen. Existerande klasser kan utvidgas med nya funktioner utan att behöva oroa sig över bakåtkompatibilitet. Alternativt kan en existerande klass också förlängas enligt en princip som kallas för arv (engelskans ”inheritance”), där en ny klass implementerar funktionaliteten från en gammal klass, och samtidigt erbjuder den nya klassen möjligheten att implementera sina egna egenskaper och metoder.

- **Underhåll**: Objektorienterad kod är oftast enklare att underhåll än exempelvis procedurell kod, eftersom att den följer någorlunda strikta kodningskonventioner, och är oftast skriven i ett mer självförklarande format. Vid situationer där utvecklare t.ex. hamnar vidareutveckla eller felsöka objektorienterad kod, så anses processen oftast vara betydligt enklare p.g.a. kodstrukturen.

- **Effektivitet**: I och med att objektorienterad programmering har utvecklats med baktanken att göra programmeringsprocessen enklare, så innebär det samtidigt att det gör den mera effektiv. Principen möjliggör en utvecklingsprocess där problem kan styckas till mindre problem, som i sin tur löser det stora problemet automatiskt. Detta innebär att programutvecklare som arbetar i team oftast har möjligheten till en arbetsprocess, där olika delar av kodbasen skilt kan utvecklas av individuella utvecklare, och först vid ett senare skede sammanslå koden till en fungerande helhet.

(Hayder 2007, 12–13)

3.2.2 Klasser

I objektorienterad programmering kan en klass anses vara en definition på en sak eller ting. Då klassen skapas bör man fundera på vilka funktioner och vilken data som måste sammankopplas till klassen, för att bygga upp en fungerade helhet. Exempelvis kunde klassen ”Användare” skapas så, att den erbjuder möjligheten till att lagra ett användarnamn, ett lösenord och ett ID, samt funktionen till att logga in och logga ut användaren. Informationen som lagras i klasserna kallas i allmänhet för *attribut*, medan klassens diverse funktioner har namnet *metod*. (Ullman 2013, 121). Kodexempel 4 innehåller ett exempel för en användarklass, utrustad med både attribut och metoder.

Kodexempel 4. Ett exempel på en PHP klass.

```php
<?php


class User {

  public function __construct($user, $pass, $id) {
    $this->userName = $user;
    $this->password = $pass;
    $this->id = $id;
  }

  public $userName;

  public $password;

  public $id;

  protected function logIn() {
    // Code
  }

  protected function logOut() {
    // Code
  }
}
```

3.2.3 Objekt

Efter att en klass har konstruerats, är följande steg att skapa en *instans* av klassen, d.v.s. ett objekt. Om det tidigare exemplet av en användarklass används, så skulle en instans av klassen i form av ett objekt kunna innehålla information som t.ex. användarnamn = ”Henrik”, Lösenord = ”12345”, och ID = ”1”. Samtidigt kan en annan instans av samma klass innehålla informationen användarnamn = ”Lisa”, Lösenord = ”54321”, och ID = ”2”. Dessa två är separata objekt instanser skapade av samma klass. Objektens struktur och utseende liknar varandra, med informationen och operationerna som körs på objekten har ingenting att göra med varandra. (Ullman 2013, 124)

3.2.4 Attribut

Attributen används i klasser som ett sätt att lagra information, och skiljer sig i själva verket inte mycket från vanliga variabler utanför klasser (Ullman 2013, 121 I det tidigare exemplet, kodexempel 4, skulle användarnamn, lösenord och ID klassificeras som attribut för användarklassen.

3.2.5 Metoder

Metoder används för att utföra arbete i en klass, och liknar väldigt mycket vanliga funktioner som befinner sig utanför klassen. Likt vanliga funktioner kan även metoder också ta in parametrar, och returnera olika värden (Ullman 2013, 121). I det tidigare exemplet, kodexempel 4, skulle ”logga in” och logga ut” klassificeras som metoder för användarklassen.

4 Utvecklingsmiljö

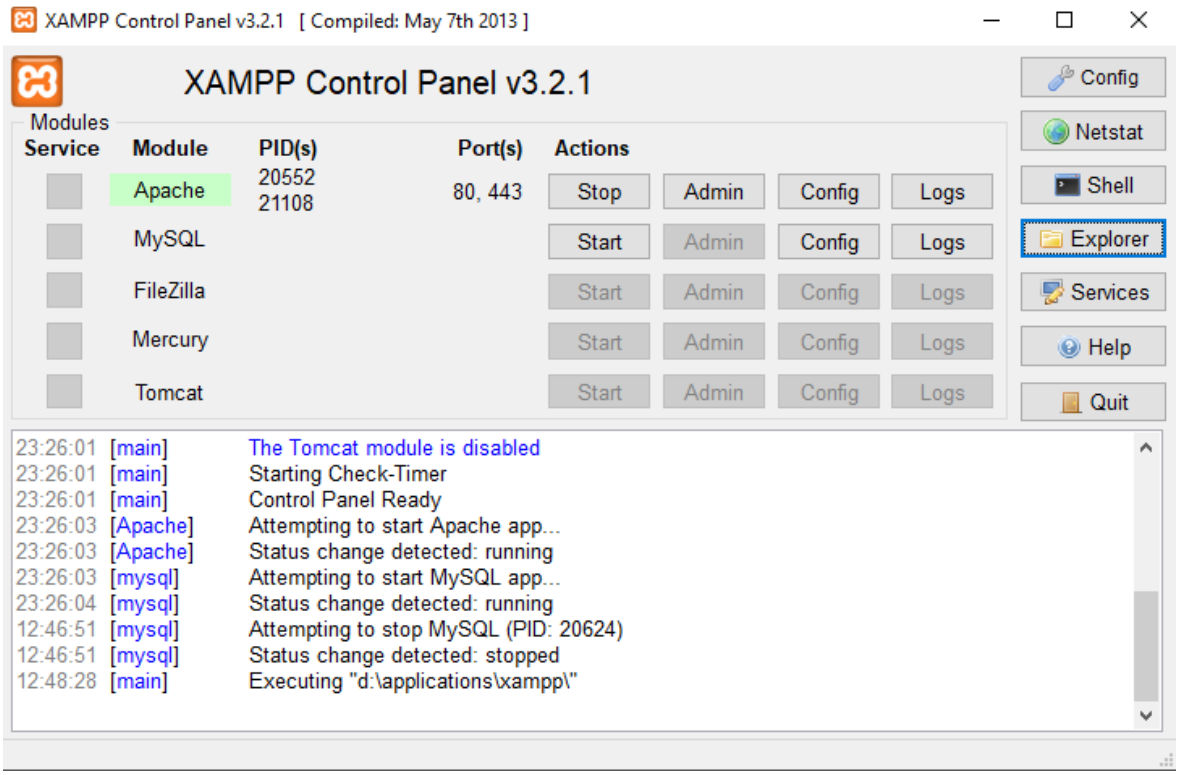
I detta kapitel kommer jag att diskutera utvecklingsmiljön, samt programvaran och tjänsterna som använts vid webbapplikationens utvecklingsskede. Eftersom att den konstruerade webbapplikationen preliminärt körs med PHP, så måste den lokala utvecklingsmiljön innehålla en webbserver som stöder PHP. En mjukvaruprodukt som kunde lösa detta problem, och som jag var bekant med sen tidigare, var Apache Friends ”XAMPP”.

Som texteditor valde jag att använda mig av GitHubs ”Atom”, eftersom att jag använt editorn sedan tidigare, och för att programmet stöder diverse tilläggspaket som t.ex. automatisk komplettering (engelskan autocomplete) för diverse programmeringsspråk.

För att hantera projektets versionshantering samt för att köra pakethanterare och JavaScript Task Runners, använde jag mig av kommandotolken ”Cmder”. Webbapplikationens källkod hanterades och lagrades på GitHubs webbhotell för versionshantering.

4.1 Xampp

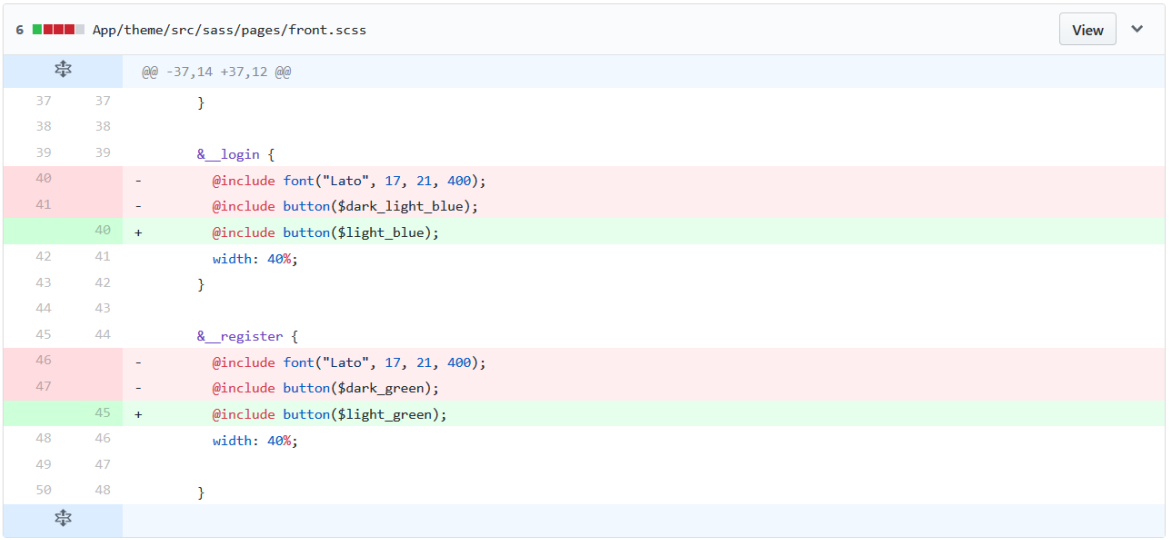
XAMPP står för ”Cross-plattform Apache MySQL PHP Perl”, och är en Apache distribution utgiven av Apache Friends. Xampp kan köras på både Windows, Mac och Linux distributioner. Programmet används till att starta upp lokala webbservrar där exempelvis PHP kod kan köras och uppkopplas till en databas. För att göra webbserverns administration enklare, erbjuder Xampp ett grafiskt användargränssnitt där diverse konfigurationsfiler kan justeras, och t.ex. Apache och MySQL kan startas och stoppas. (Dalibor 2007) Figur 4 presenterar det grafiska användargränssnittet för Xampp, där användaren presenteras möjligheten till att bl.a. starta tjänsterna för Apachae och MySQL.



Figur 4. Det grafiska användargränssnittet för XAMPP.

4.2 GitHub

GitHub kan i ett nötskal beskrivas som en webbaserad versionskontroll- och samarbetsplatform för programutvecklare. Preliminärt används GitHub till att lagra filer från mjukvaruprojekt, vars versionshantering oftast har skötts med Git. GitHub erbjuder även dess användare möjligheten att via ett grafiskt användargränssnitt få en översikt om hur diverse filer har utvecklats och förändrats under projektets gång. Då ett mjukvaruprojekt överförs till GitHub bör projektet sparas i en s.k. ”*Repository*”. Vid behov kan även andra utvecklare gör modifieringar till filerna i ett mjukvaruprojekt, som med projektägarens samtycke kan inkorporeras till det originella projektet via en s.k. ”*Pull request*”. (How-To Geek 2016). Figur 5 presenterar en bild av GitHubs översikt över filförändringar. Raderna markerade med rött är kod som blivit borttagen under den senaste gången som projektet sparades, medan de gröna raderna representerar kod som lagts till.



Figur 5. GitHubs översikt över filförändringar.

5 Planering av applikationen

I detta kapitel diskuteras applikationens olika planeringsfaser, samt hur arbetet har planerats rent praktiskt. Eftersom att webbapplikationen inte hade en beställare, så har jag ganska långt själv fått styra riktningen som applikationens utseende och funktionalitetskrav har gått i. Jag valde i ett ganska tidigt skede att applikationen skulle basera sig på någon form av CRUD (create, read, update delte) funktionalitet. Det här betyder att applikationen även kom att kräva någon from av möjlighet för dess användare att registrera sig, samt till att logga in och ut.

5.1 Funktionalitetskrav

Till applikationens mest grundläggande funktionalitet hör CRUD och användarregistrering. CRUD står för ”Create”, ”Read”, ”Update” och ”Delete,” och betyder i ett nötskal att:

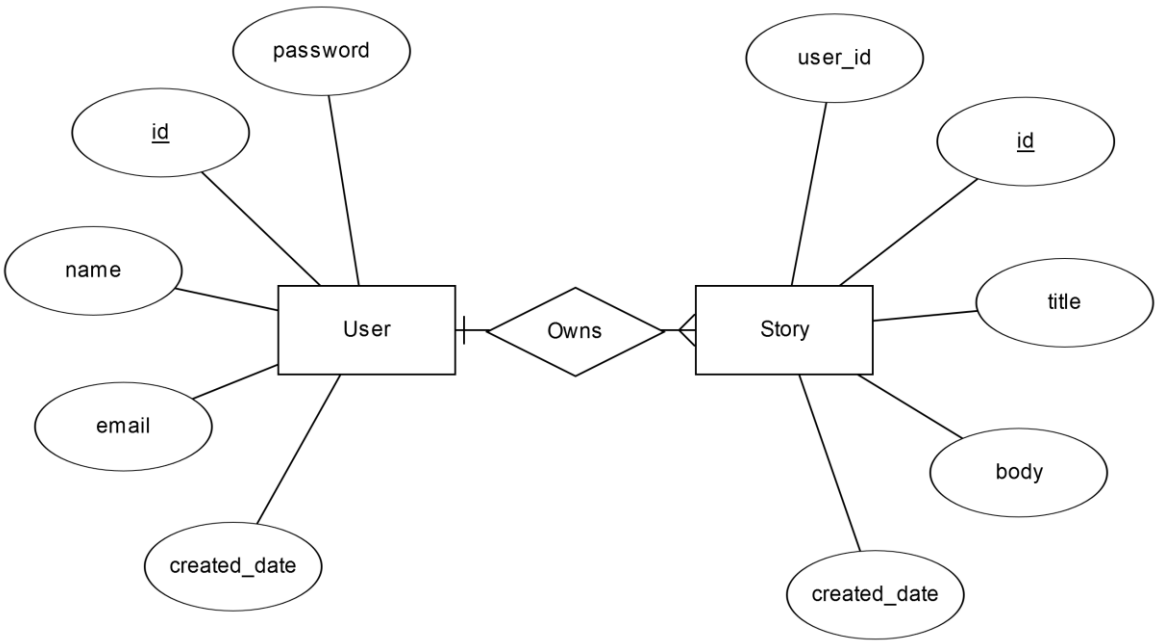
- **Create**: Applikationens användare ska ha möjligheten till att skapa nya berättelser som lagras i en databas.
- **Read**: Applikationens användare ska ha möjligheten till att läsa både sina egna och andra användares berättelser.
- **Update**: Applikationens användare ska ha möjligheten till att göra förändringar i sina egna berättelser.

- **Delete**: Applikationens användare ska ha möjligheten till att ta bort sina egna berättelser från databasen.

Till användarnas funktionalitetskrav hör även möjligheten för att kunna registrera sig, samt till att logga in och ut. De viktigaste aspekterna gällande användarregistreringen är att två (2) användare inte kan använda sig av samma registrerings-uppgifter (email), samt att deras lösenord inte får sparas som klar text i applikationens databas.

5.2 Applikationens databasstruktur

Eftersom att applikationen kommer att basera sig kring användare och berättelser, så måste två stycken tabeller skapas i applikationens databas. Tabellerna skapas med hjälp av phpMyAdmin. phpMyAdmin är ett webb-baserat verktyg där MySQL databaser kan administreras. Verktyget följer med i Xampps basinstallation. Före jag skapade tabellerna skissade jag först upp ett ER-diagram (Figur 6), vars uppgift är att beskriva de olika attributen och relationerna som tabellerna kommer att ha. Figurerna 7 och 8 är skärmbilder av de slutliga tabellerna, som implementerade med hjälp av phpMyAdmin.



Figur 6. ER-diagram över databasens tabeller.

	#	Name	Type	Collation	Attributes	Null	Default	Extra
☐	1	**id** 🔑	int(11)			No	*None*	AUTO_INCREMENT
☐	2	**user_id**	int(11)			No	*None*	
☐	3	**title**	varchar(255)	latin1_swedish_ci		No	*None*	
☐	4	**body**	text	latin1_swedish_ci		No	*None*	
☐	5	**created_date**	datetime			No	CURRENT_TIMESTAMP	

Figur 7. Story-tabellens struktur, representerad i phpMyAdmin.

	#	Name	Type	Collation	Attributes	Null	Default	Extra
☐	1	**id** 🔑	int(11)			No	*None*	AUTO_INCREMENT
☐	2	**name**	varchar(255)	latin1_swedish_ci		No	*None*	
☐	3	**email**	varchar(255)	latin1_swedish_ci		No	*None*	
☐	4	**password**	varchar(255)	latin1_swedish_ci		No	*None*	
☐	5	**created_date**	datetime			No	CURRENT_TIMESTAMP	

Figur 8. Users-tabellens struktur, representerad i phpMyAdmin.

6 Utveckling av applikationen

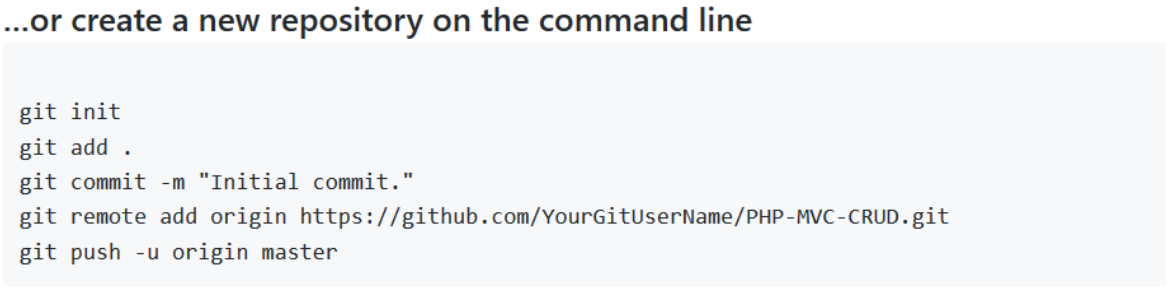
I detta kapitel diskuteras applikationens olika utvecklingsskeden; hur jag gått till väga i början av utvecklingsprocessen, och då jag tillagt ny funktionalitet till applikationen. Kapitlet diskuterar också projektets olika pakethanterare och deras installation, samt arbetsflödet gällande versionshanteringen.

6.1 Uppläggning av projektet – Git och GitHub

Eftersom applikationens kod ska lagras och hanteras på GitHubs serverar, så måste först en s.k. ”*Repository*” skapas på GitHub. Efter att en användare registrerat sig på GitHubs webbplats, är skapandet av lagringsutrymmet en relativt enkel process. Användaren erbjuds enkla och tydliga instruktioner över hur tjänsten tas i bruk, samt hur den bör används vid utvecklingsskedet av diverse projekt.

För att kunna utnyttja lagringsutrymmet på GitHub, måste vi ha ett mjukvaruprojekt vars kodfiler hanteras med något versionshanteringssystem. Eftersom jag redan tidigare hade

använt mig av versionshanteringssystemet Git, så valde jag att använda det vid utvecklingen av detta projekt också. Efter att mappen med projektets första kodfiler hade skapats, måste en uppkoppling upprättas mellan den lokala utvecklingsmiljön, och GitHubs server. Uppkopplingen samt överflyttningen av de första filerna gick enkelt med hjälp av GitHubs instruktioner, vilka beskriver uppkopplingsprocessen till GitHub med användningen av en kommandotolk. Figur 9 innehåller instruktionerna som bör följas då ett mjukvaruprojekt för första gången överförs till GitHub med hjälp av en komandtolk.



Figur 9. GitHubs instruktioner för uppkoppling via kommandotolken.

6.2 Uppläggning av projektet – Composer och NPM

Följande steg var att installera projektets två pakethanterare, Composer och NPM. Både Composer och NPM erbjuder installationsprogram för Windows baserad operativsystem, vilket gjorde installationsprocessen relativt enkel. Efter att programmen var installerade, använde jag Composer till att installera projektets *mallmotor* Twig, samt NPM till att installera projektets *task runner*, Gulp. Vid installation av tilläggspaket skapar både Composer och NPM .json baserade textfiler (Kodexempel 5), vars uppgift är att hålla reda på vilka tilläggspaket som installerats under mjukvaruprojektets utvecklingsskede. Meningen är att dessa textfiler ska inkluderas till projektets versionshantering, för att andra utvecklare vid ett senare skede skall kunna installera identiska tilläggspaket i sina egna lokala utvecklingsmiljöer.

Kodexempel 5. Exempel på NPM:s package.json, och Composers composer.json

6.3 Uppläggning av projektet – Gulp

Efter att applikationens tilläggspaket var installerade, var följande steg att ta dem i bruk. Gulp används via kommandotolken, och kör sina uppgifter genom att läsa in instruktioner från en konfigurationsfil. Det två viktigaste arbetsuppgifterna som Gulp kom att ha under detta projekt, var att konvertera .sass filer till en .css fil, samt att komprimera JavaScript.

I Gulps konfigurationsfil (Kodexempel 6) delades arbetsuppgifterna upp i två olika kategorier. Den första typen av uppgifter klassificeras som vanliga arbetsuppgifter, och används till exempel till att komprimera JavaScript filer. Den andra typen är en s.k. medföljningsuppgift (engelskans watch task), vilka i ett nötskal används till att berätta för Gulp *när* de vanliga arbetsuppgifterna ska utföras. I detta projekt användes medföljningsuppgifterna till att uppkalla de vanliga arbetsuppgifterna, alltid då det sker förändringar i diverse .sass eller .js filer.

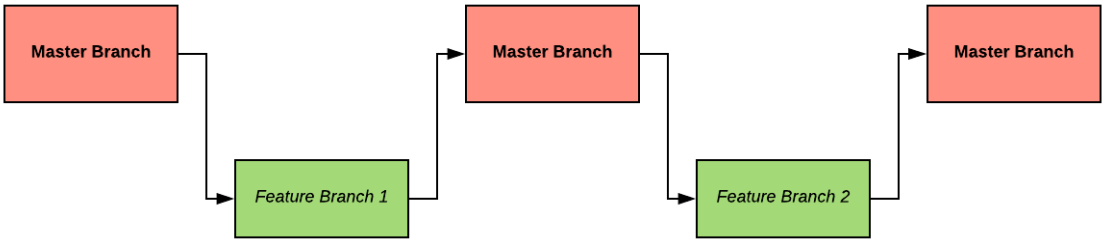
Kodexempel 6. Gulps instruktioner för att konvertera .sass filer till en .css fil.

gulpfile.js

```
var gulp = require('gulp');
var sass = require('gulp-sass');

gulp.task('default', ['sass', 'sass:watch']);

// Task for converting SCSS to CSS
gulp.task('sass', function () {
  return gulp.src('./src/sass/styles.scss')
    .pipe(sass().on('error', sass.logError))
    .pipe(gulp.dest('../../public/css'));
});

// Task for watching changes in .scss files
gulp.task('sass:watch', function () {
  gulp.watch('./src/sass/**/*.scss', ['sass']);
});
```

6.4 Versionshantering i praktiken

Efter att projektet var upplagt, var följande steg att ta i tu med versionshanteringsstrategin. Jag bestämde mig för att implementera ett system där jag alltid skapade en ny gren (engelskans branch) då jag började med utvecklingen av en ny komponent i webbapplikationen. På det här viset var det enkelt att inkapsla kodfilerna som hade med varandra att göra, och alltid då en ny funktion blev färdig, var det enkelt att inkorporera den nya egenskapen in med de gamla kodfilerna.



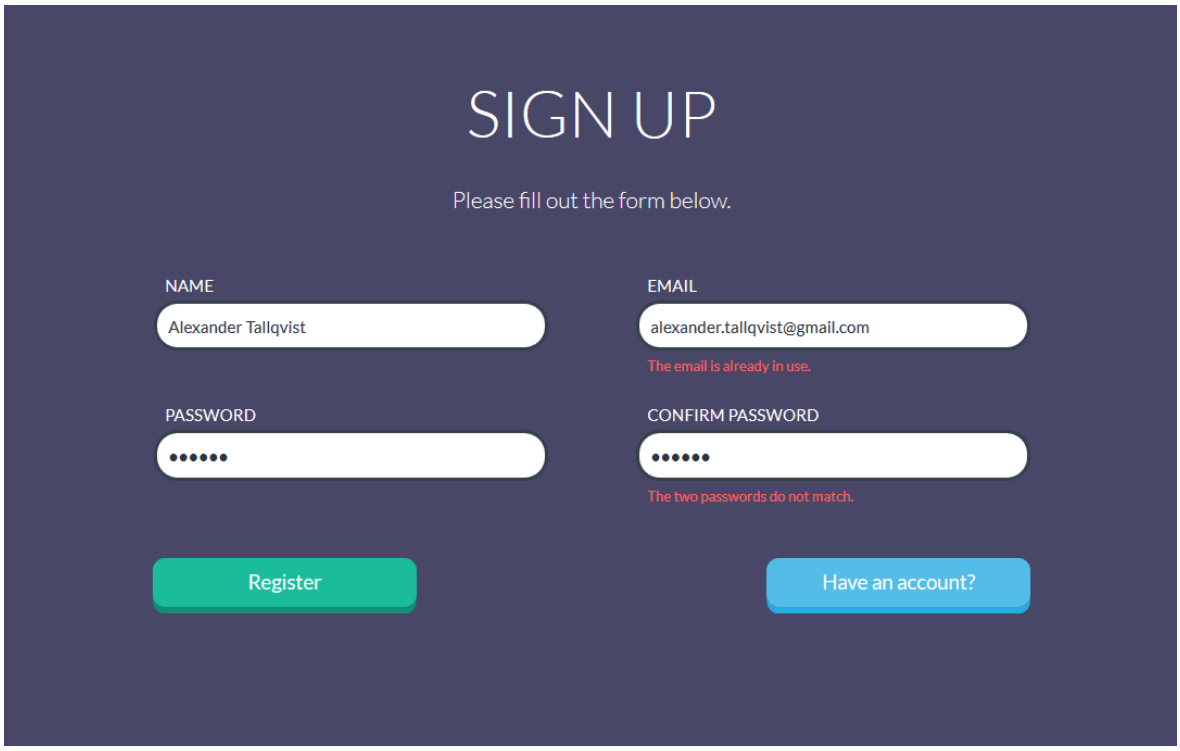
Figur 10. Exempel på projektets versionshanteringsstrategi.

7 Funktionalitet

Till den färdiga webbapplikationens funktionaliteter hör tre (3) grundegenskaper:

1) *Registrering, inloggning och utloggning*: Användarna har möjligheten att registrera sig till applikationen via ett registreringsformulär (Figur 11). Formuläret är utrustat med fyra (4) st. fält: Namn, Email, Lösenord och Lösenord på nytt. Fälten valideras på serversidan, och vid situationer där en validering misslyckas, skickas ett felmeddelande tillbaka till användaren. Till de olika valideringsmetoderna hör:

    - **Namn**, namnfältet får inte vara tomt.
    - **Email**, email fältet får inte vara tomt, och email adressen får inte existera hos en annan användare.
    - **Lösenord och Lösenord på nytt**, lösenordet måste vara minst 8 tecken långt, och de båda lösenordsfälten måste innehålla samma värden.



Figur 11. Webbapplikationens registreringsformulär med exempel på felmeddelanden.

Inloggningen till applikationen sker via ett inloggningsformulär, där användaren bes fylla i hens email och lösenord. Vid situationer där ett motsvarande email- och lösenordspar inte hittas i applikationens databas, skickas ett felmeddelande tillbaka till användaren.

2) *Skapande, redigering och raderandet av berättelser:* En inloggad användare har möjligheten att skapa nya berättelser, samt till att editera och radera sina egna berättelser. Formuläret (Figur 12) för skapningen av berättelser är utrustat med två (2) st. fält: Titel och Brödtext. Förutom titeln och brödtexten sparas även skapningstiden av berättelsen samt en koppling till användaren i form av användar-ID i applikationens databas.

   Formulärets fält valideras på serversidan, och vid situationer där en validering misslyckas, skickas ett felmeddelande tillbaka till användaren. Till de olika valideringsmetoderna hör:

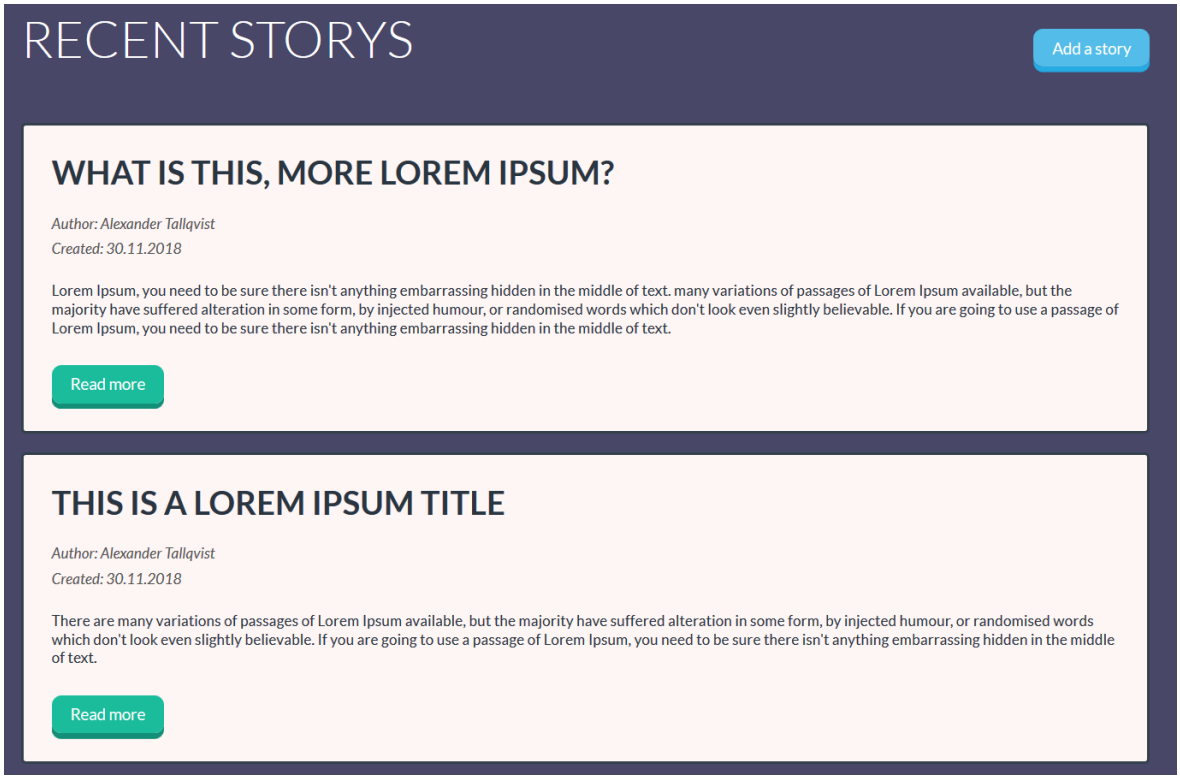
   - **Titel**, titelfältet får inte vara tomt, och inte heller längre än 150 tecken.
   - **Brödtext**, brödtexten får inte vara tom, och inte heller längre än 2000 tecken.

Figur 12. Applikationens berättelseformulär, med exempel på felmeddelanden.

Vid situationer där en användare besöker en berättelse, kör applikationen en validering över om användaren är ägaren till berättelsen som hen landar på. Då användaren visar sig vara ägaren till berättelsen, presenteras hen med möjligheten till att redigera och radera berättelsen (Figur 13). Redigeringsformuläret är identiskt formuläret för skapandet av nya berättelser, med undantaget till att fälten i redigeringsformuläret färdigt är ifyllda med informationen från berättelsen som editeras.

Figur 13. Användaren presenteras med möjligheten för redigering och radering av en berättelse.

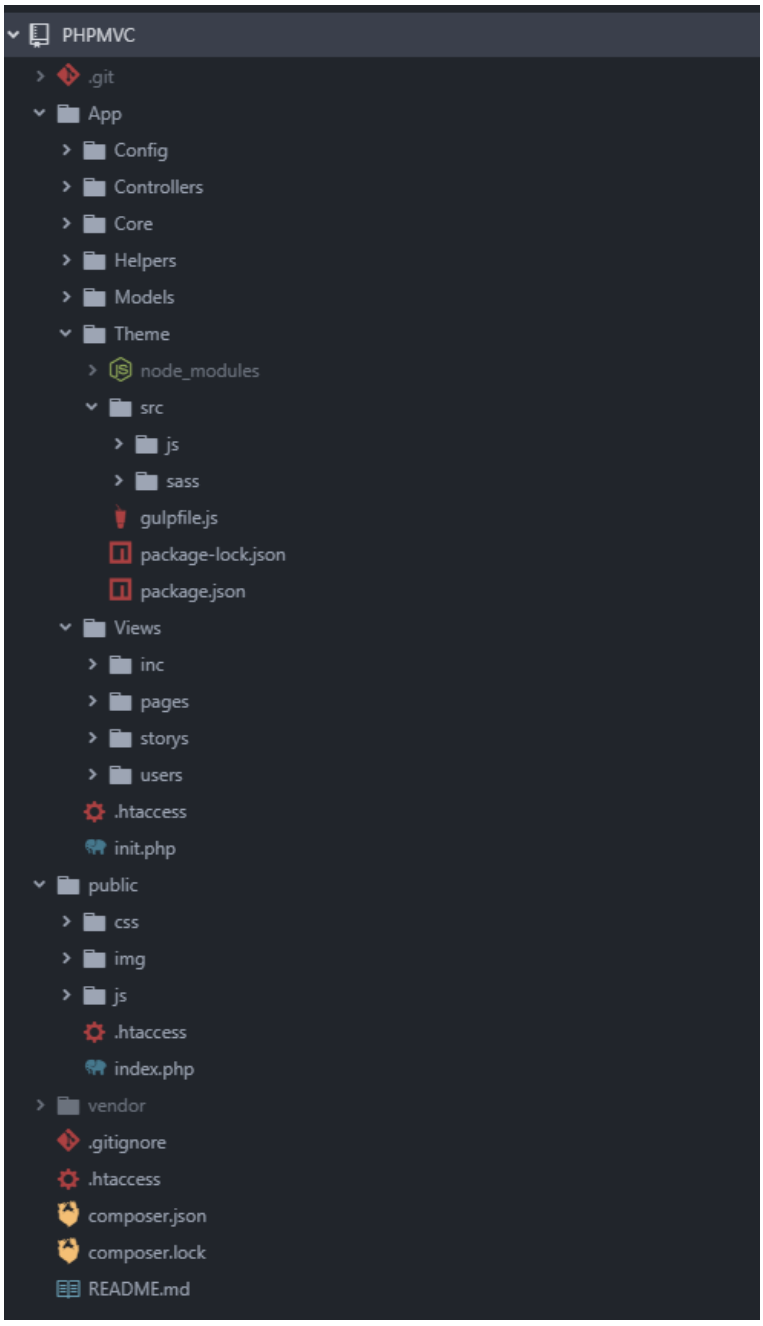
3) *Läsning av andras berättelser:* Applikationen inkluderar även en vy där användarna presenteras med en översyn över andras berättelser (Figur 14). Sidan innehåller de tio (10) senaste berättelserna i ett förkortat format, och berättelserna är utrustade med en länk där hela berättelsen kan läsas.



Figur 14. Applikationens översyn över andras berättelser.

7.1 Mappstruktur

Applikationens filer är indelade i tre (3) st. huvudmappar: App, Public och Vendor. App mappen innehåller applikationens alla PHP filer, okomprimerade CSS och JavaScript filer, samt applikationens mallar (views), som är uppbyggda med HTML och Twig. Public mappen innehåller applikationens komprimerade CSS och JavaScript filer, bilder, och en index.php fil. Alla förfrågningar som når applikationen omdirigeras till public mappen, där index filen används till att starta exekveringen av applikationen. Vendor mappen innehåller applikationens alla PHP relaterade tilläggspaket, som i början av projektet installerats med hjälp av Composer.



Figur 15. Projektets mapphierarki.

7.2 Back-end uppbyggnad

Applikationens alla back-end klasser kan indelas enligt projektets mapphierarki in i en (1) av fyra (4) olika huvudgrupper: Core, Controllers, Models eller Helpers. I detta kapitel diskuteras tanken bakom indelningen samt vad det rent konkret är som händer, då en användare besöker applikationen.

7.2.1 Back-end uppbyggnad, Core

Core mappen innehåller klasser som är utrustade med applikationens basfunktionaliteter. Till dessa klasser hör:

En databasklass, Database.php: Databasklassen är utrustad med attribut och metoder för att föra in, editera, ta bort och hämta data från applikations databas.

En basmodellklass, Model.php: Modellklassen (Kodexempel 7) använder databasklassen för att möjliggöra en uppkoppling till databasen med applikationens modeller. Modellklassen ärvs av alla andra modeller i applikationen, vilket automatiskt ger dem möjligheten till att diskutera med databasen

En routerklass, Router.php: Routerklassen används i början av applikationens exekvering till att registrera tillåtna rutter i applikationen, och till att koppla dessa rutter till controllers och deras metoder.

En viewklass, View.php: Viewklassen används av applikationens olika controllers för att skriva ut data i applikationens vyer. Viewklassen är utrustad med metoder som tillåter utskrivningen av data både i twig och i php format.

Kodexempel 7. Basmodellen i Core mappen.

Model.php

```php
<?php

/**
 * @file
 * The Main Model Class.
 *
 * @author Alexander Tallqvist <alexander.tallqvist@edu.novia.fi>
 */


namespace App\Core;

use App\Core\Database;


class Model {

  /**
   * An instance of our database connection.
   * @var PDO
   */
  protected $db;


  /**
   * The class constructor
   */
  function __construct() {
    $this->db = new Database;
  }

}

```

7.2.2 Back-end uppbyggnad, Controllers

Controllers mappen innehåller applikationens alla controllers. En controller kallas av applikationens routerklass, då applikationens användare landar på en registrerad och tillåten rutt. Då exekveringen av en controller påbörjas, registrerar de automatiskt in sina modeller i klassens konstruktormetod. Alla applikationens controllers är utrustad med metoder, som tillsammans med klassen de tillhör påkallas beroende på URL:n som en användare besöker. Exempelvis skulle till exempel controllern ”User” (Kodexempel 8) och dess metod ”logOut” kunna påkallas, då en användare besöker rutten /users/logout.

Kodexempel 8. Users controllern registrerar User modellen i klassens konstruktor metod.

Users.php

```php
<?php

/**
 * @file
 * The Users Controller.
 *
 * @author Alexander Tallqvist <alexander.tallqvist@edu.novia.fi>
 */


namespace App\Controllers;

use App\Core\View;
use App\Models\User;
use App\Helpers\Redirect;
use App\Helpers\Messages;


class Users {


  /**
   * Contains an instance of the User Model.
   * @var User
   */
  private $userModel;


  /**
   * The class constructor.
   */
  function __construct() {
    $this->userModel = new User;
  }

```

7.2.3 Back-end uppbyggnad, Models

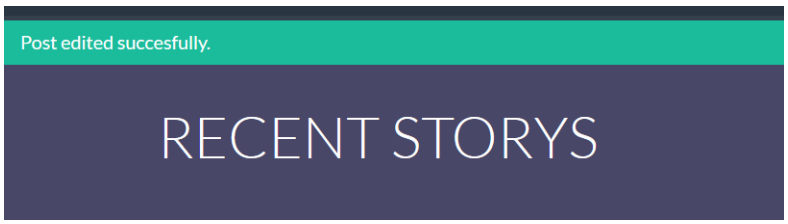
Applikationens alla modeller kan hittas i Models mappen. Modellerna används av applikationens controllers till att utföra arbete. Exempelvis skulle till exempel ”Users” controllern kunna använda ”User” modellen (Kodexempel 9) till att logga in eller logga ut en användare. Alla modellerna i applikationen ärver en basmodell från Core mappen, vilket betyder att modellerna automatiskt har möjligheten att koppla sig till en databas.

Kodexempel 9. User modellens login metod, som kallas av Users controllern då en användare försöker logga in.

```
82
83      /**
84       * The login method.
85       * A method that attempts to login a user.
86       *
87       * @param string $email
88       * The email that the user entered to the login form.
89       *
90       * @param string $password
91       * The password that the user entered to the login form.
92       *
93       * @return mixed
94       * Returns false if the login attempt failed. Returns an user
95       * array if successful.
96       */
97      public function login($email, $password) {
98        $result = self::findUserByEmail($email);
99
100       if ($result) {
101         $salt_and_hash = $result->password;
102         if (password_verify($password, $salt_and_hash)) {
103           return $result;
104         } else {
105           return false;
106         }
107       } else {
108         return false;
109       }
110     }
```

7.2.4 Back-end uppbyggnad, Helpers

Helpers mappen innehåller applikationens alla hjälpklasser. Hjälp-klasserna används preliminärt av applikationens controllers, och skulle i princip kunna ha klassificerats som modeller. Jag har dock valt att separera vissa hjälp-klasser från modellerna, eftersom jag velat bygga upp en hierarki, där en controllerklass alltid har en motsvarande modellklass. Exempelvis använder sig ”Users” controller preliminärt av ”User” modellen, och ”Storys” controller sig av ”Story” modellen. Funktionalitet som jag ansåg att inte passade in i en existerande modell implementerades då med hjälp av en hjälp-klass. Till hjälp-klassernas funktionalitet hör bl.a. ”Messages” klassen, som används till att visa upp s.k. blixtmeddelande (engelskans flash message) för applikationens användare. Figur 16 presenterar en bild på ett blixtmeddelande, där en användare informeras om att ändringarna hon gjort till sin berättelse har sparats i databasen.



Figur 16. Exempel på blixtmeddelanden (flash message).

7.2.5 Back-end uppbyggnad, Exekvering av applikationen

1) Användaren förs till index.php filen i public mappen med hjälp av .htaccess filer.

2) index.php skapar en ny instans av routerklassen (Kodexempel 10), och registrerar tillåtna rutter.

3) Användarens URL styckas upp, och routerklassen kollar om URL:n som användaren besöker har blivit registrerad.

4) Om rutten är registrerad, kallar routerklassen på ruttens controller och den önskade metoden. I andra fall slutas exekveringen av koden.

5) Controllern använder sig av diverse modeller för att uppfylla kraven som programmerats in i den uppkallade metoden.

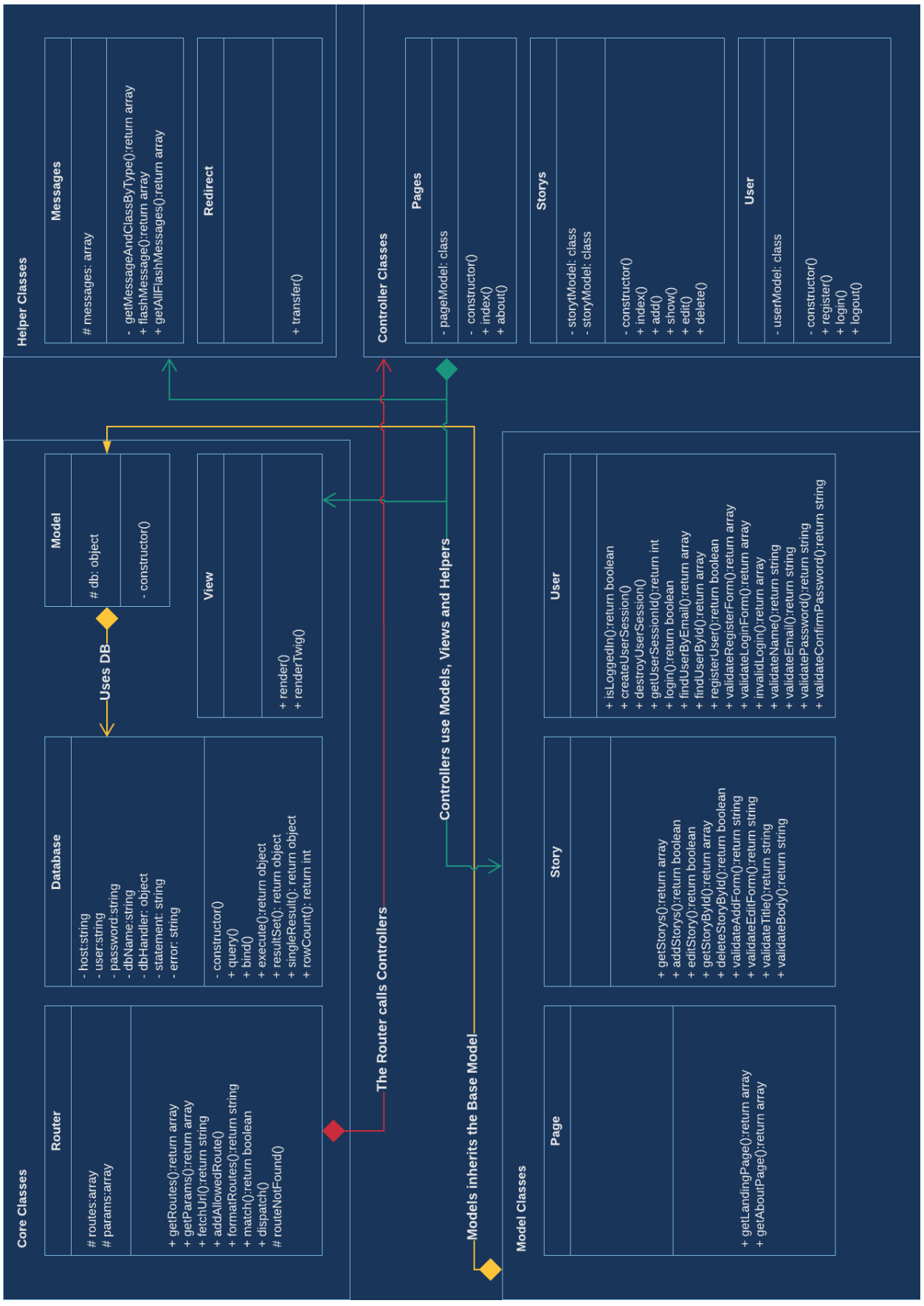
6) Modellerna utför arbete med hjälp av basmodellen och databasklassen, och strävar efter att returnerar data till controllern.

7) Controllern tar emot all data från modellen, och skickar den vidare till en view i applikationen.

8) Viewn skrivs ut till användaren med den upphämtade datan.

Kodexempel 10. Exempel på användarrutter som registreras i index.php.

```
8
9   # INITIALIZE THE CORE ROUTER
10  $route_registerer = new App\Core\Router;
11
12  # INITIALIZE ROUTES
13
14  // User routes
15  $route_registerer->addAllowedRoute('users/register', [
16                                      'controller' => 'Users',
17                                      'action' => 'register'
18                                  ]);
19
20  $route_registerer->addAllowedRoute('users/login', ['controller' => 'Users', 'action' => 'login']);
21  $route_registerer->addAllowedRoute('users/logout', ['controller' => 'Users', 'action' => 'logout']);
```



Figur 17. Projektets UML diagram.

7.3 Front-end uppbyggnad

I detta kapitel diskuteras applikationens front-end uppbyggnad samt dess vyer (templates) i allmänhet. Applikationens alla vyer ligger i ”Views” mappen, och har ansvaret att skriva ut informationen som de fått från en controller för applikationens användare. Vyerna är kategoriserade i olika undermappar (i detta fall ”includes”, ”storyes”, ”users” och ”pages”), för att enklare kunna hålla koll på vyerna vid en situation där applikationen växer.

Applikationens vyer (templates) är implementerade med Twig, och har utvecklats enligt DRY principen (engelskans don’t repeat yourself). Detta betyder att vyerna använder sig av en modell där applikationens vanligaste vyer, som t.ex. vyn för sidfoten och navigeringen, automatiskt inkluderas i applikationens andra vyer.

Applikationens stilfiler ligger i ”Theme” mappen (Figur 18), och även här har filerna uppdelade i olika undermappar, vilket i detta fall är ”general”, ”pages”, ”storys”, ”regions” och ”users”. Alla stilfiler är skrivna med hjälp av SASS och de utnyttjar både BEM och SMACSS arkitekturmodellerna i stora drag.

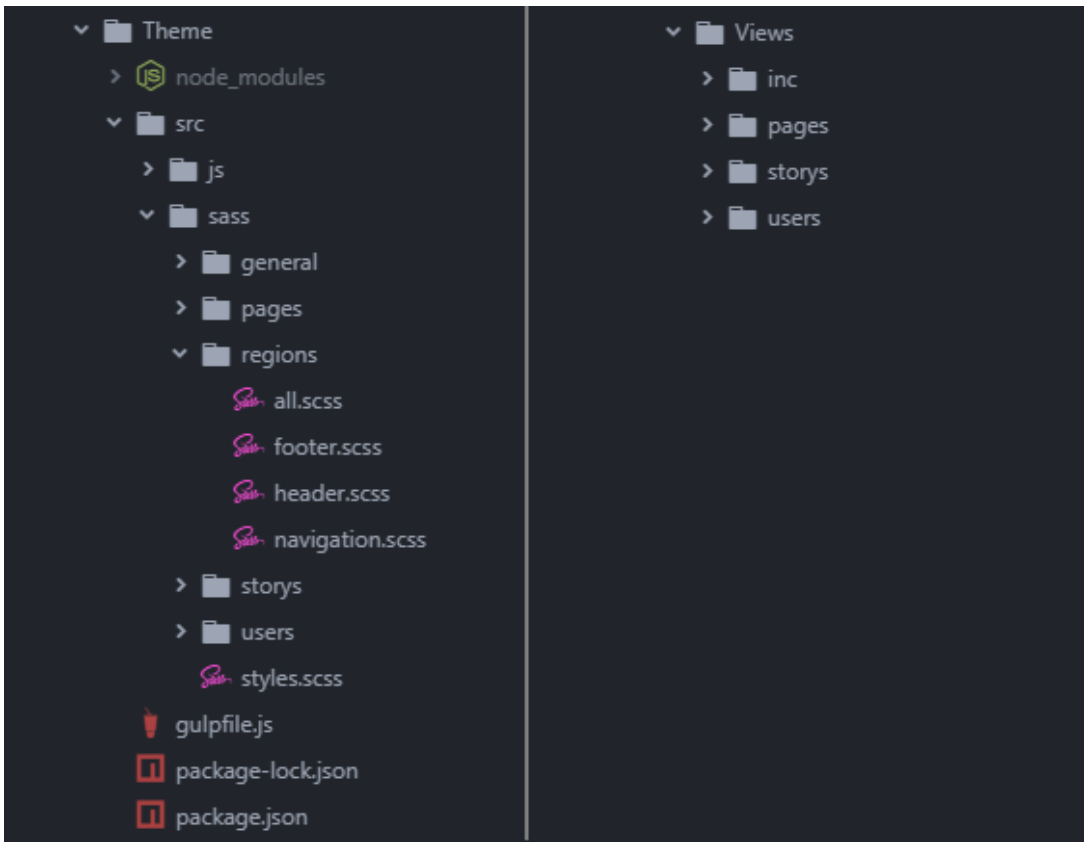
Alla applikationens front-end komponenter är i stora drag utvecklade enligt BEM principen, d.v.s. ”Block-Elemnt-Modifier”. I praktiken betyder det här att:

1. En enskild huvudkomponent (block) ges en huvudklass, exempelvis ”story”.

2. Ett element som tillhör huvudkomponenten ges en klass som prefigeras med ”—elementnamn”, exempelvis ”story—add” eller ”story—form”.

3. Underliggande element ges vidare en klass som prefigeras med ”__underelement”, exempelvis ”story—form__buttons” eller ”story—form__field-group”.

Denna uppbyggnad ger både mall- och stilfilerna (Kodexempel 11&12) en klasshierarki som åtminstone i mina ögon är mycket logisk, samt enkel att vidareutveckla. Exempelvis skull den underliggande ”Add Story” komponenten enkelt kunna omvandlas till en ”Edit Story” komponent, där grovarbetet skulle bestå av att byta klassnamnet för ”story—add” komponenten till ”story—edit”.



Figur 18. Projektets Theme och Views mappar.

Kodexempel 11. En Twigg komponent för tilläggning av nya berättelser, implementerad med BEM principen.

```
{% include  "./inc/header.html" %}

  <div class="story--add">
    <h1 class="story--add--title">Add a new Story</h1>
    <p class="story--add--description">Create a new story.</p>
    <a href="{{ constant('ROOT') }}/story" class="story--add--back">Go Back</a>
    <form class="story--add--form" action="{{ constant('URLROOT') }}/story/add" method="post">

      <div class="story--add--form__group">
        <label for="title">Title</label>
        <input type="text" name="title" value="{{ title }}" class="{{ (title_error is empty) ? 'correct' : 'invalid' }}">
        <span>{{ title_error }}</span>
      </div>

      <div class="story--add--form__group">
        <label for="body">Body</label>
        <textarea name="body" class="{{ (body_error is empty) ? 'correct' : 'invalid' }}">{{ body|raw }}</textarea>
        <span>{{ body_error }}</span>
      </div>

      <div class="story--add--form__buttons">
        <input class="story--add--form__buttons__submit" type="submit" value="Submit Story">
      </div>
    </form>
  </div>

{% include  "./inc/footer.html" %}
```

Kodexempel 12. En SASS komponent för tilläggning av nya berättelser, implementerad med BEM principen.

```
.story {

  &--add {
    width: 80%;
    margin: 0 auto;
    padding-top: 70px;
    display: flex;
    justify-content: space-between;
    flex-wrap: wrap;
    position: relative;

    &--title {…

    &--description {…

    &--back {…

    &--form {
      display: flex;
      justify-content: space-between;
      flex-wrap: wrap;
      width: 100%;

      &__group {
        @include form-group();
        margin-bottom: 30px;
        width: 100%;
      }

      &__buttons {

        &__submit {
          @include button($light_green);
          border: none;
        }
      }
    }
  }
}
```

8 Kritisk granskning

I all sin allmänhet anser jag att projektet som en helhet och den slutliga applikationen lyckats riktigt bra. Applikationen innehåller den funktionaliteten jag önskat, och jag har fått bekanta mig med nya arkitekturmodeller på både klient- och serversidan. Dessutom anser jag mig nu ha en mer grundläggande förståelse över hur skalbara webbaserade applikation bör planeras och struktureras.

Fastän jag anser projektets slutresultat vara relativ bra, så har arbetet även vissa aspekter som jag möjligtvis skulle ha kunnat förbättra. Planeringen av applikationen samt utvecklingsprocessen är troligtvis de två saker som jag skulle ha kunnat utveckla mest. Exempelvis ansåg jag det emellanåt vara svårt att bestämma mig vilken funktionalitet eller komponent som jag till följande skulle implementera i applikationen. En grundläggande plan över i vilken ordning applikationens olika delar skulle implementeras hade varit bra att ha, och är säkert någonting som jag kommer att satsa på mera vid mitt följande projekt.

Även applikationens back-end klasser skulle jag möjligtvis ha kunnat finslipa vidare och planerat bättre. Den tillfälliga implementationen använder sig av vissa hjälp-klasser som från ett MVC perspektiv i skulle ha kunnat vara modeller. Jag valde dock att bygga upp en hierarki där en controller-klass alltid hade en korresponderande modell-klass. På den nuvarande skalan fungerar detta tankesätt relativt bra, men frågan är om jag vid ett senare skede skulle ha stött på problem, om jag fortsatt separera program logiken i både hjälp- och modell-klasser.

Applikationens front-end struktur har i mina ögon lyckats relativt bra. Speciellt tycket jag om att implementera front-end komponenterna med BEM tekniken. Tack vare Twig blev applikationens vyer inte alltför uppsvällda med for-loopar och if-satser, som jag vanligtvis skulle ha hamnat implementera med PHP. Uppstyckandet av stilfilerna enligt SMACSS principen gjorde även skrivandet av applikationens SASS en enkel process – Filerna blev generellt mycket korta och lättlästa, på grund av förlängningsspråkets förmåga att återanvända kod med dess funktioner och variabler, och tack vare möjligheten att utnyttja BEM klasserna på ett logiskt sätt med hjälp av ”nesting”.

9 Avslutning

Det mest grundläggande syftet med detta arbete har varit att förbättra mina kunskaper som en webbutvecklare. Målet var att utveckla en CRUD applikation med hjälp av OOP-programmeringsmetoden samt MVC arkitekturmodellen, och samtidigt koppla in både BEM och SMACSS tankesätten vid applikationens front-end.

Den slutliga applikationen uppfyller dessa krav, och jag anser slutprodukten vara relativt lyckad. Mina allmänna kunskaper som en webbutvecklare har definitivt förbättrats; Jag anser mig nu ha en bättre förståelse över tankeprocessen och besluten som bör befattas då en webbapplikation utvecklas från ”scratch”. Dessutom anser jag mig ha en djupare förståelse över programmeringsmetoderna och arkitekturmodellerna som arbetet tangerade, både på applikationens front- och back-end.

Mina tankar angående påståendet om att dagens nya programmerare skulle avancera sig framåt i en för hastig takt, är fortfarande närvarande. Dock anser jag nu att orsaken till varför jag tidigare tvivlat på mina egna kunskaper, kan ha sina rötter i det s.k. ”bluffsyndromet” (engelskans imposter syndrome). Faktum är att webbutveckling som en bransch oftast utvecklas i en takt, som ur en utomståendes synpunkt skulle kunna betraktas som obegriplig. Ibland känns det som om det varje vecka skulle komma ut ett nytt ramverk eller verktyg, som strävar efter att underlätta webbutvecklarens liv. Dessa nya verktyg kan dock föra med sig motsatsen till den önskade effekten, eftersom att det oftast kräver en relativt stor insats av individen som vill bekanta sig med dem.

Kanske är det rent av naturligt att vi utvecklare ibland tvivlar på vårt eget kunnande och känner oss rostiga. I en bransch som är så bred som webbutveckling kan väl ingen behärska alla de olika delområdena, oberoende av insats och intresse för ämnet? Medan jag avancerar mig framåt i min karriär och mitt kunnande ökar, så hoppas jag att dessa tankar sakta men säkert vissnar bort. Tillfälligt har jag funnit mycket hopp och styrka i det engelska ordspråket: ” A jack of all trades is a master of none, but oftentimes better than a master of one.”

Till slut vill jag ännu tacka både Klaus Hansen och Kim Roos för all vägledning, kunskap och hjälp som jag inte bara fått under skrivandet av detta arbete, utan också i allmänhet, under mina studier vid Novia, Raseborg.

Källförteckning

Atlassian (u.å.a). *What is version control* [Online]
https://www.atlassian.com/git/tutorials/what-is-version-control [hämtat 13.4.2018]

Atlassian (u.å.b). *What is Git* [Online]
https://www.atlassian.com/git/tutorials/what-is-git [hämtat 13.4.2018]

BEM (u.å.). *Introduction* [Online]
http://getbem.com/introduction [hämtat 12.11.2018]

Clark, D., 2013. *Beginning C# Object-Oriented Programming, Second Edition.*
United States of America, Apress

Codecademy (u.å). *What is Node?* [Online]
https://www.codecademy.com/articles/what-is-node [hämtat 13.11.2018]

Composer (u.å.). *Dependency Manager for PHP* [Online]
https://getcomposer.org/doc/00-intro.md [hämtat 21.3.2018]

CSS-tricks, 2015. *Gulp for Beginners* [Online]
https://css-tricks.com/gulp-for-beginners [hämtat 12.11.2018]

Dalibor D., 2007. *Installing, Configuring, and Developing with XAMPP* [Online]
http://dalibor.dvorski.net/downloads/docs/InstallingConfiguringDevelopingWithXAMPP.pdf [hämtat 14.11.2018]

ECMA (u.å.). *ECMAScript 2015 Language Specification*. *Introduction* [Online]
http://www.ecma-international.org/ecma-262/6.0/index.html [hämtat 13.4.2018]

Hayder, H., 2007. *Object-Oriented Programming with PHP5.*
United Kingdom, Packt Publishing

How-To Geek, 2016. *What Is GitHub, and What Is It Used For?* [Online]
https://www.howtogeek.com/180167/htg-explains-what-is-github-and-what-do-geeks-use-it-for/ [hämtat 14.11.2018]

Ljubuncic, I., 2011. *Apache Web Server Complete Guide.* [Online]
https://www.dedoimedo.com/computers/apache_book_part.html [hämtat 28.3.2018]

MDN, 2018. *What is JavaScript?* [Online]
https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript [hämtat 13.4.2018]

Nixon, R., 2014. *Learning PHP, MySQL, JavaScript, CSS & HTML5, Third Edition.* United States of America, O´Reilly Media, Inc.

NPM (u.å.). *About npm* [Online]
https://docs.npmjs.com/about-npm [hämtat 12.11.2018]

PHP-FIG, 2018. *Frequently Asked Questions*. [Online]
https://www.php-fig.org/faqs [hämtat 21.3.2018]

Pitt, C., 2012. *Pro PHP MVC.* United States of America, Apress

Sass (u.å.). *Sass Basics* [Online]
https://sass-lang.com/guide [hämtat 13.4.2018]

SMACSS (u.å.). *Categorizing CSS Rules* [Online]
https://smacss.com/book/categorizing [hämtat 12.11.2018]

Twig (u.å). *The flexible, fast, and secure template engine for PHP* [Online]
https://twig.symfony.com/ [hämtat 13.4.2018]

Ullman, L., 2013. *PHP Advanced and Object-Oriented Programming, Third Edition.* United States of America, Pearson Education, Inc.

Ullman, L., 2018. *PHP and MySQL for Dynamic Web Sites, Fifth Edition.* United States of America, Pearson Education, Inc.

Welling, L. & Thomson, L., 2017. *PHP and MySQL Web Development, Fifth Edition.* United States of America, Pearson Education, Inc.

W3C, 2017. HTML 5.2 [Online]
https://www.w3.org/TR/html/introduction.html#introduction [hämtat 14.4.2018]

W3C (u.å.). *HTML & CSS*. *What is CSS?* [Online]
https://www.w3.org/standards/webdesign/htmlcss [hämtat 13.4.2018]

Figurförteckning

Kodförteckning