Projektuppgift

Webbutveckling III

CV

Elias Eriksson

Mittuniversitetet

MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

MITTUNIVERSITETET
Avdelningen för informationssystem och -teknologi

Författare: Elias Eriksson, eler2006@student.miun.se
Utbildningsprogram: Webbutveckling, 120 hp
Huvudområde: Datateknik
Termin, år: XX, 20XX

Sammanfattning

Ett CV i form av en webbsida byggs. CV:ts innehåll lagras i en databas som har en REST webbtjänst som gränssnitt för att interagera med databasen. REST webbtjänsten stödjer CRUD operationer som används av ett admingränssnitt för att kunna göra modifikationer till databasens innehåll.

Innehållsförteckning

Terminologi

En eventuell förteckning över termer, förkortningar och variabelnamn med korta förklaringar placeras efter innehållsförteckningen. Observera att man måste förklara begrepp och förkortningar första gången de används i den löpande texten, även om rapporten har ett terminologiavsnitt.

Akronymer/Förkortningar

REST	REpresentational State Transfer
CRUD	Create, Read, Update & Delete
ORM	Object Relational Mapping

1 Introduktion

1.1 Bakgrund och problemmotivering

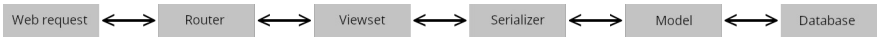
En webblösning för ett enklare CV ska skapas. Lösningen kommer att vara ett headless CMS som kommer att bestå av en REST webbtjänst, ett admingränssnitt och en publik webbplats. REST webbtjänsten kommer att agera som en gemensam datakälla för admingränssnittet och den publika webbplatsen. Webbtjänsten kommer att stödja CRUD operationer som kommer att användas av admingränssnittet för att modifiera innehållet i webbtjänstens databas. Databasens innehåll kommer sedan att läsas ut via webbtjänsten och formateras på den publika webbplatsen som kan delas med potentiell arbetsgivare som ett CV.

2 Teori

REST webbtjänsten som webbplatserna kommer använda sig av kommer vara utvecklad i programmeringsspråket python och använda sig utav tredje parts biblioteken Django och Django REST framework.

Att skapa en webbtjänst är ett sätt dela upp sin server logik och klient logik på ett sätt som gör det möjligt för flera applikationer att använda data från samma källa. I detta projekt kommer detta att göras med en REST model som innebär att HTTP metoder mappas till en CRUD operation t.ex. en POST request kommer att leda till att data skapas i en viss tabell. [12]

Django är ett ramverk för att utveckla webbplatser med och kommer med en inbyggd ORM som låter utvecklare definiera en klass som sedan översätts till en tabell i en databas [1]. Dessa klasser kallas för models och förutom att skapa tabeller går så är det möjligt att via dessa klasser göra CRUD operationer på databasen [2]. Django REST framework utnyttjar Djangos models för att veta vad som finns i databasen och gör det sedan möjligt att implementera en Serializer. En serializer är en annan typ av klass som gör det möjligt att översätta en Django model till bland annat JSON samt att översätta JSON tillbaka till en Django model [3]. Denna serializer klass kan sedan kan sedan användas av en viewset klass som definierar hur datan ska skickas till mottagaren samt kollar om användaren har tillåtelse för en viss CRUD operation [4]. Dessa viewsets mappas sedan till URL endpoints med en Router klass [5].



Figur 1: Kommunikationsvägen från en webb request i Django applikationen till databasen för rest webbtjänsten.

Webbtjänsten kommer att kräva admin privilegier för Create, Update och Delete operationer så om en användare försöker göra ett POST anrop utan privilegier kommer det det bli stop i ett viewset. För att göra ett anrop med privilegier används Django REST frameworks token autentisering. För att komma över en token görs en POST request till en specificerad endpoint med användarnamn och lösenord. Om användarnamnet och lösenordet matchar skickas en genererad token tillbaka i responsen. Om användaren sedan inkluderar headern `Authorization` med värdet `Token <token>` där <token> är en given token. Kommer användaren att ha ut ökat sina privilegier till att inkludera Create, Update och Delete. [6]

Admin gränssnittet är utvecklat i en kombination av PHP, TypeScript och sass. PHP används framförallt för att lättare länka mellan undersidor och inkluderingar av HTML snippets medans TypeScript används för att driva all trafik mellan REST webbtjänsten samt utskrift av innehållet som tas emot från webbtjänsten. Flera nya ES funktioner utnyttjas i TypeScript. Framförallt fetch API för att hantera webbtrafiken men också async / await för att lättare hålla koden plattare.

För att använda TypeScript på klient sidan måste TypeScript kompileras till JavaScript vilket görs med TypeScripts kompilator tsc som kan installeras via npm och köras med node.js. [9]

Eftersom webbläsare inte heller har stöd för att direkt köra sass kod så behöver sass kod först kompileras ner till CSS innan en webbläsare kan använda det. Sass gör det möjligt att skriva CSS med flera features vilket gör att det blir lättare att hålla struktur i sin källkod. [13]

Fetch API som är baserat på ES6 Promises jämfört med XMLHttpRequest som är baserat på eventlyssnare gör det lättare att göra requests. När en request görs så går det att använda await för att få JavaScript att vänta på att anropet är klart eller så kan metoden .then användas för att ge JavaScript en callback funktion. Await är smidigt att använda om allt efterkommande beror på vad ett promise värde i slutändan då det håller koden plattare och JavaScript pausar vid ett await tills promiset är fullfilled [7]. Om det däremot är av intresse att saker ska gå så parallellt som möjligt så är .then smidigt att använda då den givna callback funktionen blir schemalagd till att köras när promiset är fullfilled [8].

Den publika webbplatsen består inte av någon backend kod så dess funktionalitet är helt driven med JavaScript kompilerad från TypeScript och stylingen kommer att göras med sass som sedan kompileras ner till CSS. Denna webbplats använder sig också av fetch API samt async / await för att hämta ut data från webbtjänsten men eftersom sidan endast ska läsa ut data och därmed bara använda GET så behöver användaren inte logga in då webbtjänsten tillåter Read utan att vara autentiserad.

Den publika webbplatsen kommer att utvecklas med ett verktyg som heter gulp. Gulp hjälper till med att automatisera processer som att kompilera TypeScript till JavaScript och sass till CSS. Gulp installeras med hjälp av npm och det finns möjlighet för massor av tillägg som också finns att hämta via npm. [14]

Git kommer också att utnyttjas i detta projekt för att spara versionshistorik både lokalt och på en server. Git är ett verktyg som låter utvecklare spara snapshots över hur ett projekt såg ut. När en snapshot är sparad är det möjligt att senare gå tillbaka ifall något gjordes som inte fungerade. [10] Alla dessa snapshots sparas i i ett en mapp som kallas repository och kan laddas upp på en server som en backup och möjliggör att andra utvecklare kan ladda ner projektet och göra ändringar och ladda upp. [11]

3 Metod

Det fösta som gjordes va att börja skissa över hur REST webbtjänstens databas skulle se ut. Eftersom att Django används skapas en hel del tabeller i databasen automatiskt men det är bara Djangos auth_user tabell som används. Denna tabell innehåller information om registrerade användare med användarnamn och lösenord. Denna tabell används av Django REST frameworks token autentisering för att binda en token till en existerande användare. Eftersom att alla användare kommer att anses vara admin kommer inget mer behöva vara direkt kopplat till Djangos auth_user tabell. Det kommer även att behövas en tabell för varje typ av innehåll som sidan ska ha stöd för. I detta fall behövs en tabell för kurser, jobb och webbplatser. (8) Webbtjänsten stödjer pageing för att göra så att det inte krävs att användaren hämtar alla data som potentiellt skulle kunna vara mycket.

Designskisser skapades över admin gränssnittets sida. (1) Iden är att ha en hopdragen lista för varje typ av innehåll och när användaren trycker på en knapp för att expandera listan eller trycker på knappen för att lägga till innehåll så expanderas listan. Om användaren tryckte på lägga till innehåll visas ett nytt innehåll högst upp med tomma fält. Listan som expanderats innehåller allt innehåll från en viss tabell i webbtjänstens databas. Utöver möjligheten att fylla i inputs med innehåll så finns det några knappar på höger sida. En knapp för att flytta innehållet högre upp i listan och en knapp för att flytta innehåll ner i listan. Tanken är att vissa saker kanske är mer relevanta än andra saker och de mer relevanta sakerna kan prioriteras att läggas högre upp i listan och detta kommer att reflekteras sen på den publika webbplatsen. Utöver en upp och ner knapp finns det även en radera knapp som raderar innehållet och en ångra knapp som ändrar tillbaka innehållet till det som är lagrat i databasen. När önskade ändringar är gjorde i input fälten och / eller något tagits bort i listan kommer en bekräfta knapp lysa upp grönt i listans meny där som applicerar alla ändringar gjorda över allt innehåll i listan till REST webbtjänstens databas. Om det är något som går fel med en ändring så ska användaren bli ner scrollad till det innehåll som gav ett fel och ett felmeddelande ska skrivas ut. (4)

Det gjordes också skisser över den publika webbplatsen. (2) Eftersom att tabeller ofta är svårt att mobilanpassas så gjordes en skiss över hur tabellen kommer att se ut i mobil vy också. (3) När användaren först kommer in på sidan så kommer det bara att finnas tre knappar på sidan och vardera knapp representerar en typ av innehåll från REST webbtjänstens databas. Om användaren trycker på en knapp hämtas en sida med innehåll från REST webbtjänsten som presenteras i en tabell form. Om det finns mer än en sida med innehåll så kommer en paginator att visas på sidan där användaren kan navigera mellan dom. (5)

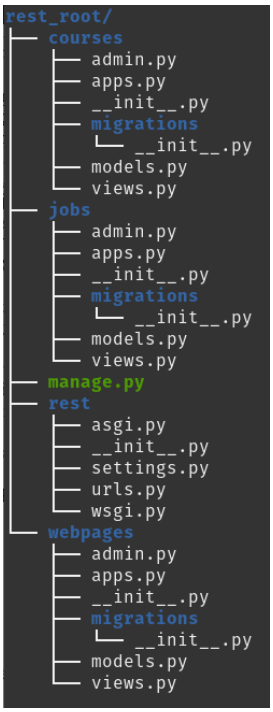
4 Konstruktion

4.1 REST

Först skapades alla designskisser enligt tidigare beskrivningar i GIMP och ett ER diagram över databasen i https://draw.io. Efter detta började jobb på REST webbtjänsten enligt ER diagrammet.

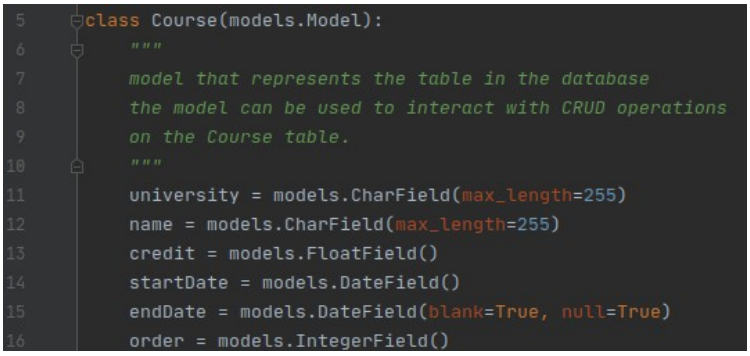
Ett nytt repo skapades på github och klonades sedan ner med `git clone https://github.com/EliasEriksson/miun-web3-mom6-rest`. I denna mapp skapades ett nytt python virtual environment med `python3 -m venv venv` och aktiverades sen med `source venv/bin/activate`. Django och Django REST framework, Django cors headers, markdown och django-filter installeras sedan med `python -m pip install django djangorestframewor django-cors-headers django-filter markdown` och en requirements fil skapas med `python -m pip freeze > requirements.txt` för att sedan lätt kunna installera alla nödvändinga paket med `python -m pip install -r requirements.txt`.

Efter att alla paket installeras skapas ett nytt django projekt med `python -m django startproject rest` som genererar ett nytt django projekt i en mapp rest. Namnet på rest mappen ändras till rest_root för att lättare skilja på mapparna i projektet då det även finns en mapp inuti som heter rest. Tre nya django applikationer skapas sedan med `python rest_root/manage.py startapp courses`, `python rest_root/manage.py startapp jobs` och `python rest_root/manage.py startapp webpages`. Vardera applikations test fil tas bort då den inte kommer användas. (Figur 2)



Figur 2: Mappstrukturen efter skapandet av ett nytt Django projekt och 3 appar lagts till (tests.py är bort tagna).

Med alla Django applikationer uppsatta så modellerades förs kurs modellen genom att redigera rest_root/courses/models.py där en ny klass Courses skapas som ärver från Djangos model klass. I denna klass definieras några attribut och vars värde är någon typ av Djangos model fält. Fälten university, name, credit, startData, endData och order definieras med ett lämpligt fält. (Figur 3) Sedan skapas en Jobs klass i rest_root/jobs/models.py med attributen company, title, startDate, endDate och order. Till sista skapas en en model WebPages med attributen title, description, url, och order. Både Courses och Jobs endDate tillåts vara null för att kunna representera att det inte finns ett slutdatum för att visa att kursen eller jobbet är nuvarande.



```
class Course(models.Model):
    """
    model that represents the table in the database
    the model can be used to interact with CRUD operations
    on the Course table.
    """
    university = models.CharField(max_length=255)
    name = models.CharField(max_length=255)
    credit = models.FloatField()
    startDate = models.DateField()
    endDate = models.DateField(blank=True, null=True)
    order = models.IntegerField()
```

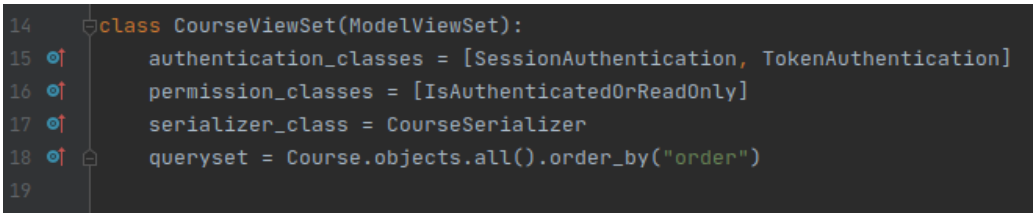
Figur 3: En model klass som representerar ett table med fälten university, name, credit, startDate, endDate och order.

Till varje model skapas en serializer klass. Serializern skapas genom att ärva från Django REST frameworks HyperLinekdModelSerializer klass. Utöver att att ärva så definieras även en inre klass i serializern som heter Meta. Meta klassen innehåller meta information och där definieras vilken model det är serializern ska använda sig av samt vilka fält. (Figur 4)

```
class CourseSerializer(serializers.HyperlinkedModelSerializer):
    class Meta:
        model = Course
        fields = ["id", "university", "name", "credit", "startDate", "endDate", "order"]
```

Figur 4: En serializer klass för Kurs modellen.

Till varje serializer skapas sedan en ViewSet klass som ärver från Django REST frameworks ModelViewSet klass. I denna klass definieras hur en användare kan bli autentiserad, vilka behörigheter som gäller, vilken serializer som ska användas och hur datan ska efterfrågas från databasen. En användare anses vara tillräckligt autentiserad om om användaren loggat in och har en live session eller om användaren har en token. Behörigheterna sätts till att kräva inloggning annars så är tjänsten satt till att endast tillåta att bli läst med GET requests. Eftersom det görs en ViewSet klass till vardera serializer klass så kommer serializern va satt till en av vardera. All data ska kunna hämtas ut från tjänsten så det definieras men också att den ska vara sorterat enligt order. (Figur 5)



```
class CourseViewSet(ModelViewSet):
    authentication_classes = [SessionAuthentication, TokenAuthentication]
    permission_classes = [IsAuthenticatedOrReadOnly]
    serializer_class = CourseSerializer
    queryset = Course.objects.all().order_by("order")
```

Figur 5: En ViewSet klass för kurs modellens serializer.

Med alla klasser definierade redigeras rest_root/rest/urls.py till att importera alla ViewSet klasser definierade ovan och en ny Router skapas. Till routern registreras

en endpointsen courses, jobs och webpages och vardera endpoint mappas till sin respektive ViewSet klass. Routerns url patterns läggs till i variabeln urlpatterns. Till urlpatterns läggs också en path till endpointen token/ som kommer göra det möjligt för användare att få en token.

```
24  # create a router that registers the endpoints for the resources
25  router = DefaultRouter()
26  router.register("courses", CourseViewSet)
27  router.register("jobs", JobViewSet)
28  router.register("webpages", WebPageViewSet)
29
30
31  urlpatterns = [
32      path('admin/', admin.site.urls),  # page for django admin
33      path("token/", obtain_auth_token),  # page for token access
34      path("auth/", include("rest_framework.urls")),  # non admin login portal
35  ] + router.urls  # adds all the URLs from the router
```

Figur 6: En Router skapas och alla urler läggs till till urlpatterns variabeln.

Inställningarna i rest_root/rest/settings.py ändras till att inkludera alla apparna som skapats och alla tredje parts bibliotek som installerats. Cors headern ändras med genom att modifiera variabeln CORS_ORIGIN_ALLOW_ALL till True vilket ger värdet * i headern. De tillåtna HTTP metoderna definieras i variabeln CORS_ALLOW_METHODS till DELETE, GET; OPTIONS, PATCH, POST och PUT. Paging sätts på genom att modifiera Django REST frameworks inställningsvariabel REST_FRAMEWORK och en standard sida sätts till att vara tio. Databas inställningar och Djangos SECRET_KEY (som används till kryptering) läses från filen rest_root/rest/credentials.json och värdena lästa därifrån appliceras till variablerna DATABASES samt SECRET_KEY. ALLOWED_HOST ändras till * och STATIC_ROOT defineras till mappen rest_root/static/.

Alla tabeller som behövs skapas sedan med `python rest_root/manage.py makemigrations` och `python rest_root/manage.py migrate`. Sedan kan en användare skapas med `python rest_root/manage.py createsuperuser`.

4.2 Admin

Innan sidan börjar byggas skapas en config.php fil. Denna config innehåller två variabler som definierar vilken mapp som är roten till sidan via webben samt vart roten till projektet är på webbserverns filsystem. rootURL variabeln blir roten för en sökväg via webben och rootPath blir rotsökvägen på filsystemet. Detta utnyttjas för att lätt länka till filer som ligger i undermappar genom att prefixa alla sökvägar med antingen rootURL eller rootPath beroende på om det är något användaren ska efterfråga eller om det är något som php utnyttjar på filsystemet.

En inloggnings sida byggs sedan. Inloggnings sidan består av ett enkelt form som efterfrågar användarnamn och lösenord. Om formet skickas kommer JavaScript att överskrida standard beteendet av ett form och istället skicka en POST request till webbtjänstens /token/ endpoint. Om användarnamnet och lösenordet är korrekt kommer en token att skickas tillbaka från servern. Denna token sparas i webbläsarens local storage med JavaScript och användaren kommer blir

omdirigerad till startsidan. Om uppgifterna som angavs inte fungerar skrivs istället ett felmeddelande ut till användaren. (6)

När inloggningen fungerar byggs startsidan. Om användaren går in på startsidan och det inte finns en token i local storage kommer användaren att omdirigerad till inloggnings sidan. Om en token finns kommer istället sidan att börja ladda in innehåll från webbtjänsten. Alla typer av innehåll hämtas samtidigt från webbtjänsten. Hämtandet av data görs i två påföljande requests för vardera endpoint. Först så hämtas så mycket data som webbtjänstens pager är satt till att ge vilket är tio resultat. (6) Med denna data kommer också metadata om hur mycket det finns lagrat. Detta utnyttjas för att hämta ut resterande data från endpointen genom att lägga till GET parametrarna offset och limit. Offset sätts till så mycket data som först hämtades vilket är tio och limit sätts till så mycket data det finns att hämta - offset.

Efter att alla data har hämtats används den för att initialisera en klass Content. Content klassen klassen styr hur innehållet ska renderas till HTML och vilka eventlyssnare som ska läggas på. En eventlyssnare för vardera input läggs på för att uppdatera ett internt objekt som innehåller nuvarande data. Content har koll på hur datan såg ut när den först hämtades vilket gör det möjligt att se om det finns en skillnad i datan som den är skriven i DOM.

Om datan i DOM blir ändrad eller om innehållets ordning blir ändrad av att användaren trycker på upp eller ner knappen så kommer en PUT request att förberedas och lagras i en lista. (6)

Om en användare trycker på plus knappen kommer ett nytt innehåll dyka upp högst upp i listan och användaren har möjlighet att fylla i den data den vill. När det nya innehållet genereras kommer en POST request att förberedas. (6)

En användare har möjlighet att radera ett innehåll genom att trycka på kryss knappen. Om kryssknappen blir tryckt kommer den eventuellt förberedda PUT eller POST requesten göras om till en DELETE request. (6)

Om det är en skillnad på datan i DOM och original datan så lyser en undo knapp upp i DOM. Vid tryck av undo knappen återställs datan i DOM till den data som finns lagrad som original data. (6)

Om datan är ändrad i något Content objekt så kommer även checkbox knappen att lysa upp. Vid tryck av checkbox knappen kommer alla förberedda requests att skickas till webbtjänsten. PUT eller POST request går igenom kommer deras original data att uppdateras till den data som skickades. Om en DELETE request gick igenom kommer dess Content objekt att raderas. Om det blir en error kommer ett felmeddelande skrivas ut till DOM och den första errorn att sparas temporärt tills alla requests är klara och användaren kommer bli ner scrollad till till innehållet som gav error. (6)

Sidan stylades sedan med sass. Ett färgtema valdes först ut till ett mörkt tema och sedan användes sass invert, lighten och darken funktioner för att omvandla de mörka färgerna automatiskt till en ljus motsvarighet. Sedan användes media

queryn prefered color scheme för att avgöra vilka färger som ska visas för användaren.

4.3 Publik webbplats

Innan den publika webbplatsen börjar byggas installeras gulp, gulp-typescript, gulp-sass, gulp-cssnano, gulp-terser och gulp-browsersync med npm. Sedan skrivs en gulpfil som vid uppstart kompilerar all TypeScript till JavaScript och som minimerar den kompilerade Javascriptkoden, kompilerar all Sass till CSS och som minimerar den kompilerade CSSkoden samt så startas en utvecklings server som refreshar webbplatsen varje gång Sass eller TypeScript kompileras.

När användaren först gå in på sidan händer inget speciellt. Användaren måste först trycka på en utav knapparna ”Arbetslivserfarenheter”, ”Utbildning” eller ”Projekt”. När användaren har tryckt på en knapp kommer en sida med relaterat innehåll att hämtas från webbtjänsten. Om det finns mer än en sida att hämta kommer en paginator att renderas på sidan där användaren kan välja vilken sida som ska visas. (7) För att hämta en annan sida läggs GET parametrarna offset och limit till till requesten och värdet på offset styrs med vilken sida användaren vill till.

När innehållet renderas så är det stylat som en tabell vid bredare skärmar. Om däremot skärmstorleken minskar kommer tabellhuvudet att gömmas med display: none och visas framför vardera värde istället.

Sidan följer samma färgtema som admin sidan och samma teknik med färgerna utnyttjas där för att avgöra vilket färgtema som ska visas för användaren.

5 Resultat

Restwebbtjänsten som skapats stödjer alla CRUD funktioner, är öppet tillgänglig via https://web3mom6rest.eliaseriksson.io/. Tjänsten tillåter anslutning från vilken källa som helst då cross origin headern är satt till *. Tjänsten är även lösenordsskyddad så inga andra förutom admins kan göra ändringar i databasen. För att ändra på innehållet används admingränssnittet som tvingar en användare att logga in innan det är möjligt att göra ändringar i webbtjänstens databas. Admingränssnittets UI är gjord på ett sådant sätt att det är möjligt att göra alla ändringar man vill och sedan när allt är klart kan allt skickas till webbtjänsten samtidigt. Innehållet kan sedan visas i snygga tabeller på den publika webbsidan.

6 Slutsatser

Eftersom att det nu va ett krav att använda gulp så har det gjort för klient delen av projektet. Men det känns inte som att gulp gör något speciellt för mig då jag inte vill köra funktioner som terser och cssmin under utvecklings stadiet då det gör det mycket jobbigare att använda debuggern i webbläsaren. För mig räcker det att bara använda Jetbrains inbyggda file watcher för att göra det som gulp gör för mig. Om det används mycket bilder så kan det säkert vara smidigt men om bilder inte används så känns det som en onödig dependency under utvecklingen då tid måste läggas ner på att skriva gulp filen. Jämfört med att bara slå på mina fördefinierade file watchers i min IDE.

Större delen av tiden med lösningen lades ner på att göra admingränssnittet interaktivt med knappar som ändrar status när något händer, animerat scroll etc. Alla denna funktionalitet gav grund till mycket buggar som tog tid att lösa så det fanns ingen tid att omstrukturera admingränssnittets kod som för närvarande är ganska spretig och onödigt lång. Admingränssnittets kod skulle egentligen behöva omstruktureras. Det finns även möjlighet att förbättra klient koden.

Det som bland annat skulle kunna förbättras är:

- Lägga admin sidans generella inputs i global scope och de klasserna som specialiseras hade inte behövts då det skulle gå att avgöra vilka inputs som ska användas beroende på vilken endpoint som blir specificerad i xyzContentManager.create() (där xyz anvädns för att visa att det är en av de klasser som ärver från ContentManager). Detta skulle kunna ta bort alla .then metoder som ligger efter .create() metoden samt att inga specialiseringsklasser som ärver från ContentManager skulle behövas.

- Toggle klassen för admin sidan är mer eller mindre onödig och ska förmodligen hanteras i ContentManager klassen istället.

- På samma sätt som ContentManagers kedjade .then metod efter .create() skulle kunna tas bort skulle klientens Loader.create() kedjade .then metod också kunna tas bort och skötas någon annan stans för att göra det lättare och mer intuitivt att initialisera klassen.

- När alla reqeusts skickas i ContentManager.syncRequest() används en for loop med await. Det det skulle gå snabbare om alla requests skickades samtidigt med Promise.allSettled istället som exempel.

- Hur olika eventlyssnare defineras i Content.render() och hur de kallar på specifika metoder från ContentManager måste kunna skrivas om på ett bättre sätt för hur det är gjort nu är väldigt spretigt och är inte skalbart på något sätt. Att försöka härma en eventlyssnare funkar skulle kanske funka.

Källförteckning

[1] Django Software Foundation ”Migrations” https://docs.djangoproject.com/en/3.2/topics/migrations/ hämtad 2021-10-30.

[2] Django Software Foundation ”Making queries” https://docs.djangoproject.com/en/3.2/topics/db/queries/ hämtad 2021-10-30.

[3] DRF ”Serializers” https://www.django-rest-framework.org/api-guide/serializers/ hämtad 2021-10-30.

[4] DRF ”ViewSets” https://www.django-rest-framework.org/api-guide/viewsets/ hämtad 2021-10-30.

[5] DRF ”Routers” https://www.django-rest-framework.org/api-guide/routers/ hämtad 2021-10-30.

[6] DRF ”TokenAuthentication” https://www.django-rest-framework.org/api-guide/authentication/#tokenauthentication hämtad 2021-10-30.

[7] MDN ”Making asynchronous programming easier with async and await” https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await hämtad 2021-10-30.

[8] MDN ”Promise.prototype.then()” https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then hämtad 2021-10-30.

[9] MDN ”TypeScript: optional static typing for JavaScript” https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/Svelte_TypeScript#typescript_optional_static_typing_for_javascript hämtad 2021-10-30.

[10] git-scm ”A3.3 Appendix C: Git Commands - Basic Snapshotting” https://git-scm.com/book/en/v2/Appendix-C%3A-Git-Commands-Basic-Snapshotting hämtad 2021-10-30.

[11] GitHub ”About repositories” https://docs.github.com/en/repositories/creating-and-managing-repositories/about-repositories hämtad 2021-10-30.

[12] Malin Larsson ””REST-webbtjänster med PHP
https://play.miun.se/media/rest_webbservices_dt173g_ht20/0_aj4gprtl
Publicerad 2020-10-01 hämtad 2021-10-30.

[13] Sass ”Sass Basics” https://sass-lang.com/guide hämtad 2021-10-30.

[14] gulp ”gulp” https://www.npmjs.com/package/gulp hämtad 2021-10-30.

7 Bilagor

(1) bilagor/Admin.png

(2) bilagor/clientDesktop.png

(3) bilagor/clientMobile.png

(4) bilagor/adminStory.png

(5) bilagor/clientStory.png

(6) bilagor/adminFlowchart.png

(7) bilagor/clientFlowchart.png

(8) bilagor/database.png