Projektuppgift

Javascriptbaserad webbutveckling

Anarchy Cooking

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: HT, 2022

Sammanfattning

I detta projekt skapas en REST webbtjänst för att hantera recept samt ett användargränssnitt i form av en webbplats. Webbplatsen har stöd för att visa, skapa, redigera och ta bort recept. Ett recept kan enkelt skapas och redigeras med en UI byggd med React. Webbtjänsten har även stöd för att tagga recept för att göra sökningen av recept lättare vid användning utav sidans sökfunktion.

Innehållsförteckning

Introduktion

1.1 Bakgrund och problemmotivering

En hemsida för recept ska byggas. Receptsidan ska ha stöd för att visa, lägga till, redigera och ta bort recept. Det ska även gå att tagga recepten med valfria taggar och söka efter recept där det går att söka efter specifika taggar. Recepten som hittas visas i en lista och det ska gå att visa recepten ett och ett. För att skapa recept ska en titel, beskrivning, ingredienser, instruktioner samt användar taggar anges. Detta ska sedan gå att redigera och radera.

1.2 Avgränsningar

Det går att förfina sökfunktionen väldigt mycket men så fort den fungerar tillräckligt bra så kommer ingen extra tid att läggas ner på att göra den bättre och bättre.

2 Teori

2.1 Node.js

Node.js är en JavaScript runtime som använder samma Javascript motorn v8. Javascript brukar oftast exekveras i webbläsaren men Node.js öppnar upp för möjligheten att skriva applikationer som sedan kan köras i terminalen vilket gör det möjligt att skriva webbapplikationers back-end i JavaScript. [1]

2.2 Express

Express är ett ramverk för Node.js som gör det lättare att hantera webbanslutningar. Med express går det definiera specifika URL endpoints och om användaren ansluter med samma endpoint kommer koden för den endpointen att köras. Express har stöd för de olika HTTP verben vilket gör det väldigt rätt att bygga en REST API med express. [1]

2.3 MongoDB

MongoDB är en databas server men till skillnad från MySQL/MariaDB så är MongoDB en NoSQL databas. Med NoSQL menas det att databasen antingen inte huvudsakligen använder sig utav relationer och att datan lagras annorlunda jämfört med en relationsdatabas. MongoDB sparar allting i databasen som ett dokument i en samling. Varje dokument blir tilldelat ett slumpat id som är garanterat att vara unikt. I detta dokument går det att spara samma datatyper som JSON har stöd för vilket betyder att det även finns stöd för objekt och array typer till skillnad från en relationsdatabas som skulle representera exempelvis arrayer med en relation till en annan tabell. [2]

2.4 Mongoose

Mongoose är ett bibliotek som kan användas med Node.js för att lättare jobba mot MongoDB. Med Mongoose går det att i våran Node.js applikation bygga upp en datamodell med hjälp av ett schema. Schemat specificerar vilken data som existerar på modellen. När en modell är byggd går det att utföra queries mot Mongo databasen genom att använda redan färdiga metoder som automatiskt läggs till på modellen av Mongoose. [3]

2.5 Mongoose-paginate-v2

Mongoose-paginate-v2 är en plugin till Mongoose som gör det lätt att skapa pages. Där en Mongoose query skulle returnerat en array av resultat kommer istället ett objekt att returneras. Detta objekt har ett attribut docs som är en array där resultaten nu finns istället. Hur många och vilka resultat som finns med beror på vilka inställningar som skickas med vid en page query. Det styrs av en limit och page parameter. Med limit menas att resultaten kommer ut i sidor av exempelvis 10 och page styr vilken sida av 10 som hämtas. Objektet har även information om vilken limit och page som används samt om det finns en tidigare sida och nästa sida.[4]

2.6 React

React är ett front-end ramverk utvecklat av Meta (Facebook). React marknadsför sig som ett ramverk som gör det lättare att utveckla användargränssnitt. React tillåter utvecklare att bygga komponenter. En komponent är en byggsten av en hemsida och har instruktioner för hur HTML ska renderas. Detta görs via Reacts egna syntax som heter JSX. JSX ser nästan exakt ut som HTML med fördelarna att det går att skriva in referenser till variabler. En komponent kan också använda sig av något som kallas för state. State är en variabel som håller någon form av data och när state uppdateras med setState metoden som React kommer med så kommer komponenten att renderas igen. Det gör att det blir lätt att hålla det som syns på webbsidan i synk med den data som lagras i state. [5]

2.7 React-router

Ramverket React kommer inte med ett inbyggt sätt att kunna länka till undersidor. För att göra detta används ramverket React-router som är byggt för att fungera tillsammans med React. React router lägger till några nya komponenter som gör att det är möjligt att definiera routes till undersidor. En route definieras med en path och den komponenten som ska renderas på den pathen. [6]

3 Metod

För att skapa sidan kommer MongoDB att användas som databas. För att göra det lättare att arbeta med MongoDB med Javascript kommer Mongoose att användas för att definiera models.

Eftersom att taggar och ingredienser ska vara sökbara så skapas ett eget schema och modell (collection) till taggar och ingredienser. I Mongoose schemat specificeras taggarnas och ingrediensernas namn att vara unika så det inte finns två taggar med samma namn i databasen.

Recept schemat och modellen byggs sedan upp att innehålla en titel, beskrivning, ingredienser, instruktioner och taggar. Titeln och beskrivningen specificeras till strings i scheman. Ingredienserna specificeras till att vara en array utav objekt som har ett objectID till en ingrediens, hur mycket det ska vara av ingrediensen och enheten det mäts i. Instruktionerna är en array av objekt med strängar. Instruktionerna sätts till en array av objekt istället för av objekt av strängar då det kommer att underlätta att få unika nycklar till varje instruktion för React som kräver en unik nyckel för varje element som skrivs ut från en array. Receptets taggar är en array utav objekt med ett objectID till en tagg. Taggarna görs till ett objekt framförallt för att de andra två är arrayer av objekt och det håller det konsekvent.

För att sedan skapa restwebbtjänsten används express tillsammans med Mongoose och för att ge webbtjänsten paginerings stöd används Mongoose-paginate-v2.

För att skapa klient webbsidan används React för att bygga sidan. För att kommunalisera med webbtjänsten används fetch. Eftersom React kräver unika nycklar för element i arrayer så hämtas paketet uuid från npm för att underlätta med skapandet av nycklar där det inte finns ett databas id. För att starta React projektet används `npx create-react-app client --template typescript`. När sidan är färdigbyggd används `npm run build` för att bygga sidan till bara HTML, CSS och JS.

Klient sidan i sig har fyra undersidor. En startsida där recept listas ut utan ingredienser. En sida för ett specifikt recept där det går att läsa receptet i sin helhet samt en sida för redigera och skapa ett nytt recept som ser mer eller mindre lika dana ut med ett formulär där det går att lägga till och ta bort saker från receptet. (1)

För att undvika att hämta allt material på en gång på startsidan används pagination för att hämta ut innehåll i stötar. Mer innehåll hämtas när användaren scrollar ner på sidan och ser en laddnings ikon. (1)

4 Konstruktion

4.1 Back-end med express

4.1.1 Databas strukturen med Mongoose

Det första som görs är att skapa de Mongoose modeller som krävs för att spara all nödvändig data till applikationen. Hemsidan handlar om recept så en modell Recipes skapas som håller attributen title, description, ingredients, instructions och tags. Title och deskription kommer att vara av typen string medans de andra kommer vara olika array typer.

Ingredients är något som önskas lagra unikt i databasen då ingredienserna önskas vara sökbara. En model skapas därför för Ingredients med ett attribut ingredient av typen string och Recipes ingredients attribut kommer att vara en array av objekt där varje objekt består av attributen ingredient, amount, och unit. Ingredient attributen kommer att vara ett MongoDB objectID som motsvarar en ingredients objectID, amount kommer vara en float och unit kommer vara en enum av olika typer. De typerna som kommer stödjas är: L, dL, cL, mL, kg, g och st.

Instructions arrayen i Recipe modellen kommer att bestå av objekt med ett attribut instruction. En instruction representeras som ett objekt istället för en sträng då MongoDB kommer att generera ett id å instruktionen vilket kommer göra det lättare sen med React då unika keys krävs för array innehåll.

Tags är också något som önskas vara unikt och sökbart i databasen så det skapas även en model Tags. Precis som Ingredients kommer denna modell bara ha ett attribut tag av typen string och Recipes modellen kommer att vara en array utav objekt med ett attribut tag som kommer vara av typen objetID som motsvarar en tags objekt id.

På varje models attribut tilläggs kravet required och alla attribut som nämnts ska vara unika får kravet unique i sin modell. Modellen Recipes extendas sedan som till en klass där några extra metoder läggs till och några av modellens inbyggda metoder skrivs över. Varje model får även alternativet strict: throw för att se till att det inte finns några attribut utöver de attributen specificerade i schemat. Om extra data hittas kastas ett error. VersionKey: false används också för att slippa skicka med extra data onödig för detta projekt.

En statisk metod capitalize läggs till som gör allting i en sträng till lowercase om det inte är första bokstaven i strängen eller första bokstaven efter en ., ? eller ! Då görs bokstaven om till en stor bokstav. Denna metod används sedan i en till statisk metod format som tar ett objekt med samma attribut som Modellens schema. Objektets title, description och alla instruktioner blir sedan processade med metoden capitalize.

Med Dessa nya funktioner skrivs Mongoose modellens inbyggda funktion över med. Den nya metoden kallar på superklassens validate metod (den överskrivna metodens original implementation) men sedan körs även format metoden på objektet själv för att formatera title, description och instructions. På samma sätt så skrivs metoden save över som kallar på den nya metoden validate innan original implementationen av save körs.

Den nya save metoden ser till att all text har korrekta versaler men detta måste även göras vid en uppdatering av objektet. Därför skapas en ny statisk metod findAndUpdate som först formaterar datan och sen använder sig av Mongoose modellens findByIdAndUpdate. Eftersom att findByIdAndUpdate är statisk går den inte att skriva över för att få samma namn.

Recipes får även en sök metod. Sök metoden tar en sök strängs parameter och options som skickas bara passas ner i de queries som kommer göras. Sök strängen söks igenom med två olika regex mönster. Det första mönstret: /(?<=#)([\w-]+)/gu hittar allt som är sammanhängande bokstäver och bindestreck som kommer efter #. Det andra mönstret: /(?:^|\s)(\w+)/gu hittar de ord som inte det första mönstret hittar. Om mönstret som söker efter taggar matchar och får ut ord kommer orden att användas för att hämta taggarna ur databasen. De orden som inte va taggar kommer användas för att hämta ut ingredienser från databasen. Resultaten från dessa två queries kommer användas för att bygga upp tre olika sökkategorier. En relevant sökning, en titel sökning och en ingrediens sökning.

För en relevant sökning används de funna taggarna, de funna ingredienserna och sen söks även receptets titel för de ord som även användes för att hitta ingredienser. Om denna sökning ger resultat är det troligt att det är det som användaren söker efter då båda tagg, ingrediens och titeln matchar på något sätt.

För titelsökningen används bara de hittade taggarna och sen används orden för att söka i titeln. Denna sökning är inte lika restriktiv då en ingrediens inte måste finnas i titeln för att hittas här. Sökningen kräver dock fortfarande att alla funna taggar är inkluderade i receptet som hittas.

För ingrediens sökningen används de funna taggarna och de ingredienser som hittats. Denna sökning kommer vara användbar om användaren vill kunna söka på recept med specifika ingredienser och det är inget krav på att ingrediensen finns i titeln. Taggarna kommer fortfarande att begränsa sökningen till de taggar som hittats. Alla dessa tre sökningar kommer att köras med an paginerad sökning med de alternativ som skickas med i search funktionen, dessa paginerade resultat kommer returneras som ett objekt med attributen relevantSearch, titleSearch och ingredientSearch.

4.1.2 Endpoints

Eftersom att express endast kommer användas till några få endpoints används inte express generator utan all kod läggs i en fil. Högst upp i filen definieras en konstant rootURL som är den URL som hemsidan kommer att ha när sidan hostas. Konstanten apiRoot skapas också som är rot URLen plus /api som kommer att definiera startpunkten för alla endpoints.

En hjälp metod paginateParams skapas också som söker efter get parametrarna limit och page i en inkommande anslutning och returnerar ett objekt med attributen page och limit. Om get parametern page eller limit saknas finns kommer ett standardvärde användas för den parameter som saknas.

Varje model kommer att ha fullt stöd för CRUD och det kommer därför vara möjligt att med get hämta flera eller ett specifikt objekt beroende på om ett id specificeras i URLen, med post skapa ett nytt objekt, med put kan ett redan existerande objekt uppdateras om ett id specificeras och med delete kan ett specifikt objekt raderas. Varje Model kommer att ha stöd för dessa fem funktioner via fem olika endpoints. Eftersom det finns tre olika modeller så kommer 15 endpoints att skapas. Motsvarande funktioner för varje endpoint ser väldigt lika ut och Ingredients modellens fem funktioner kommer se exakt likadana ut som Tags modellens fem funktioner.

När en get request för flera objekt tas emot så kontrolleras det om det finns en get parameter s finns med. Om den finns med kommer en variabel pattern att sättas till samma värde annars kommer patterns värde sättas till en tom sträng. Sedan kontrolleras även om get parametern exact finns med i anslutningen. Om den finns med kommer pattern att bli prefixad med ^ och suffixad med $. Denna pattern variabel görs sedan om till ett regular expression som används för att söka i databasen. Sökningen blir även paginerad och hjälpmetoden paginateParams används för att få ut en limit och page till paginationen. Det som sedan hittas med sökningen kommer att skickas tillbaka till användaren med status 200.

om get requesten även innehåller ett specifikt id till en tagg eller ingrediens kommer det specifika id:t hämtas från databasen och skickas till användaren med status 200.

Vid en post request för ingredienser och taggar så görs ett försök att skicka ner hela request bodyn i Modellens konstruktor och sedan använda save metoden. Om ingen error sker i save metoden så hade den skickade bodyn i requesten korrekt format och det skapade objektet skickas tillbaka till användaren med en status 201. Om däremot en error sker i save metoden skickas errors tillbaka till användaren med status 400.

Vid en put request för ingredienser och taggar skickas request bodyn ner i modellens findByIdAndUpdate metod tillsammans med det id som är specificerat i put requesten. Om ett objekt hittas så returneras det till användaren med status 200 annars så skickas errors tillbaka med status 400.

Vid en delete request för ingredienser och taggar skickas det id som är specificerat i delete requesten i modellens findByIdAndDelete. Det objekt som togs bort skickas sedan tillbaka till användaren med status 200. Om Id:t inte finns så skickas istället en error tillbaka med status 400.

De olika endpointsen för Recipe ser lite annorlunda ut jämfört med de andra två modellerna då Recipe har id referenser till andra objekt dessa id utvecklas till sina objekt genom att använda modell metoden populate, populate söker och hämtar efter ett objekt på en specificerad databas path. Detta görs på ingrediens pathen

och tagg pathen i Recipes modellen så att användaren får ut ingrediensernas och taggens namn i samma request. Exakt samma metodik används för get (specifik), post, put och delete som tidigare för ingredienserna och taggarna med dessa populate metoder läggs till.

För Recipes get (flera) endpoint skiljer sig dock mycket från de andra. Först kommer ett objekt att byggas upp med alla alternativ som ges till en paginerad sökning att byggas upp. Sen kontrolleras om en söksträng finns med som get parameter. Om den finns med så kommer en sökning med hjälp av Recipes search metod att utföras och alternativen för pagineringen skickas med som parameter. Om ingen sök sträng finns med kommer ingen specificerad sökning göras och databasen kommer hämta det antal recept från databasen som är specificerad av get parametern limit. Resultatet skickas sedan till användaren med status 200.

4.2 Front-end med React

För att starta upp ett React projekt körs `npx create-react-app client --tempalte typescript` för att skapa en mapp client i samma projekt rot som express serverns projekt. I client mappen hamnar alla filer nödvändiga för att starta ett React projekt. Två inställningar som läggs till i den package.json fil som skapas under client är proxy: ”localhost:8083” och homepage: ”https://miun.eliaseriksson.io/jsweb/moment5/”.

Proxy inställningen styr vart någonstans React kommer skicka fetch anrop på utveckling servern. Som standard skickas requests som fetch(”/jsweb/moment5/api/recipes”) till samma origin och eftersom React körs på en annan port än vad REST webbtjänsten gör kommer anropen skickas till fel host utan proxy inställningen.

Homepage inställningen krävs när React projektet byggs till att bara vara HTML, CSS och JavaScript filer som sedan läggs upp på en server. Alla länkar mellan statiska filer kommer att prefixas med homepage URLen så att all CSS / JavaScript hittas från HTML filen.

Projektet börjar skrivas i den medföljande index.ts filen. I roten av allt som renderas skapas en HashRouter som kommer så React-router vet på vilket sätt den ska skapa länkar mellan undersidorna. Renderas en Header komponent, en main tagg med Routes och en Footer komponent. I main taggen definieras alla sidans undersidor med Route komponenter wrappade runt en Routes komponent. Varje route får en path och vilken komponent som ska renderas på den specificerade pathen. De paths som definieras är `/` där en Home komponent kommer renderas `/recipes/:id/` som renderar ViewRecipe, `/recipes/edit/:id/` som renderar EditRecipes och `/recipes/new/` som renderar NewRecipe.

Header komponenten innehåller bara sidans header som har en bild på sidans logotyp tillsammans med ett h1 element som är sidans namn. Logotypen är en Link komponent som länkar till / vid tryck.

Footer komponenten innehåller en footer tagg med text att sidan är skapad av Elias Eriksson samt en länk till sidans GitHub repository.

Home komponenten använder sig att ett state för att kunna omrendera sidan. Home komponentens state har en variabel för sök, en bool variabel för att hålla koll på om allt innehåll från APIn har hämtats och en array med recept data.

Komponenten kommer att rendera ett form med en textinput som blir kopplat till komponentens sök state, en div där all recept data skickas ner som props till en komponent RecipeSummary och en snurrande ladd symbol som kommer att hämta mer innehåll när den syns på skärmen. Det finns även en länk som länkar till `/recipes/new/`.

När sidan laddas kommer ladd symbolen att synas och sidan kommer då att göra ett anrop till APIn. En kontroll kommer att göras för att se om det finns någon sökning i state vilket det inte kommer vara vid första sidladdning så ett anrop kommer att skickas till `/recipes/?page=1`. Page vid sidladdning kommer att ha värdet 1 och kommer att uppdateras med det som APIn svarar med är nästa sida. Ett antal recept kommer att komma med svaret och data kommer att lagras i states recept array. Detta kommer att pågå så länge som variabeln fetchMore har värdet true. fetchMore får värdet true när laddningssymbolen är renderad. Och får värdet false när laddningsdonen försvinner ur bild. På detta sätt hämtas mer data bara när användaren scrollat igenom det som redan hämtats. Om allt innehåll har hämtats kommer laddningsdonen att sluta renderas. Eftersom att state ändras under hämtningen av data kommer React att rendera om komponenten och div elementet med recepten kommer att fyllas på med nya recept.

Om en sökning skulle göras på sidan kommer det trigga ett event. Detta event kommer att återställa alla variabler kopplade till hämtning av data samt allt state utom sökströngen. Detta kommer leda till att laddningssymbolen hamnar i bild och ett anrop till APIn kommer att ske. Denna gång kommer ett anrop att gå till `/recipes/?page=1&s=` där get parametern s får värdet av det som användaren har skrivit i sökrutan. Eftersom att en sökning kommer tillbaka som tre olika paginerade resultat kommer alla resultat att läggas till i states recept data array kommer relevanta sökresultat läggas till först följt av titel sökning följt av ingrediens sökning.

Varje recept kommer i Home komponenten att renderas som RecipeSummary komponenter. En RecipeSummary komponent är en div med innehållet titel som ett h2 element, beskrivningen som ett p element och receptets taggar i en till div. Titeln är även wrappad med en Link tagg som länkar `/recipes/:id/` där :id byts ut mot receptets egna id.

Om användaren navigerar till ett recepts sida kommer React-router att rendera komponenter ViewRecipe. ViewRecipe kommer att rendera receptet i sin helhet för användaren att läsa. Det finns även två länkar på sidan. En länk kommer att länka till `/` så användaren kan navigera tillbaka och en länk som navigerar till `/recipes/edit/:id`.

Om användaren navigerar från ett recept till receptets redigerings sida kommer komponenten EditRecipe att renderas. EditRecipe. EditRecipe kommer att rendera en länk tillbaka till receptets sida samt komponenten RecipeForm.

RecipeForm komponenten tar en valfri property _id. Om _id ges så kommer RecipeForm att rendera ett form med all data från receptet med det specificerade id:t. Om inget id anges kommer RecipeForm rendera ett tomt recept som används när användaren ska skapa ett nytt recept. Formet som renderas utav RecipeForm kommer i sin tur att rendera fem komponenter. En komponent för titel, beskrivning, ingredienser, instruktioner och taggar. Varje komponent kommer att få tillgång till RecipeForms state i form av att skickas med som props och vardera komponent ansvarar för att modifiera varsin del av RecipeForms state.

När formuläret blir skickat kommer först nya ingredienser som inte redan finns i databasen skapas genom API endpointen `/api/ingredients/` med post requests. Samma sak kommer att göras med taggarna för att skapa taggar som inte redan finns genom att skicka post requests till endpointen `/api/tags/`. Detta görs för att få databasen att generera id:n till de nya ingredienserna och taggarna som sedan kan användas för att göra en request med recept datan. Om RecipeForm komponenten fick ett recept id med som en prop kommer en PUT requests att skickas till `/api/recipes/:id/` annars kommer en POST request att skickas till `/api/recipes/`.

Om användaren har gått in på redigera sidan kommer formuläret även att ha en röd radera knapp. Om denna knapp blir tryckt på kommer en delete request att skickas till `/api/recipes/:id` .

RecipeForms komponenter Title och Description kommer att rendera två olika textareas som endast kommer att uppdatera RecipeForms state med ny titel och beskrivning när användaren skriver något.

Ingredients komponenten är lite mer komplex och består av en div som innehåller inputs med receptets alla nuvarande ingredienser samt en radera knapp. Ingrediensens namn hanteras av komponenten IngredientDataList som består av en text input och en datalist. När användaren skriver något i text inputen kommer en get request att skickas till `/api/ingredients/?s=` där s är det som användaren hittills har skrivit i inputen. Resultaten kommer att skrivas till datalistan vilket kommer att ge användaren känslan av en kontinuerlig autocomplete när den skriver. När användaren slutar skriva i inputen och inputen ger av ett blur event kommer en request att skickas till `/api/ingredients/?s=&exact=true` där s är det som stod i inputen när användaren lämnade den. En exakt sökning kommer att göras i databasen som bara kommer kunna ha ett resultat. Om sökningen ger en matchning kommer ingrediensens id att skrivas in i RecipeForms state. Annars kommer id:t att lämnas blankt vilket kommer indikera en ny ingrediens som. Ingredients komponenten har även en knapp som tillåter användaren att lägga till en ny ingrediens.

Instructions komponenten representeras av en div innehållande alla receptets instruktioner. Dessa instruktioner har en textarea som användaren kan redigera samt en uppåt pil, neråt pil och en delete knapp. Om användaren redigerar text

arean kommer RecipeFroms state att uppdateras med den nya texten. Om användaren trycker på uppåt pilen kommer ett recept att flyttas upp i arrayen av instruktioner och RecipeForms state kommer att uppdateras. Trycker användaren neråt pilen kommer instruktionen flyttas ner i arrayen och RecipeForms state uppdateras. Om användaren trycker på radera kommer instruktionen tas bort från arrayen och RecipeForms state kommer att uppdateras. Användaren har även möjlighet att lägga till en instruktion i listan.

Tags komponenten fungerar likt Ingredients komponenten. Tag komponenten representeras av en div som innehåller en lista av alla receptets taggar där varje tagg är en komponent TagDataList samt en radera knapp. TagDataList är en text input med en datalist. Skriver användaren något i text inputen kommer en get request att skickas till `/api/tags/?s=` där s är det användaren hittills har skrivit i text inputen. Resultaten skrivs till datalistan och när användaren lämnar inputen och ett blur event sker kommer en get request till `/api/tags/?s=&exact=true` att skickas om sökningen ger resultat kommer taggens id att läggas läggas till annars kommer id att vara tomt vilket indikerar en ny tagg. Användaren kan trycka på taggens knapp för att ta bort en existerande tagg eller trycka på knappen för att lägga till för att lägga till en ny tagg till receptet.

4.3 Host

För att hosta express servern skapas en service fil och express filerna läggs upp på egen server. Service filen läggs till som en Systemd service och startas. Nginx konfigureras till att göra en reverse proxy för inkommande webbtrafik på miun.eliaseriksson.io/jsweb/moment5/api/ till localhost:8083 som är den port som express servern körs på.

För att hosta front end filerna körs `npm run build` för att bygga React appen till att bara bestå av HTML, CSS och JS. Dessa filer läggs sedan upp på servern och Nginx konfigureras att serva filerna på urlen miun.eliaseriksson.io/jsweb/moment5/.

5 Resultat

En hemsida för recept är skapad. Lösningen har fullt CRUD stöd före recept. Det är möjligt att lägga till valfria taggar på recepten och taggarna går att söka efter på startsidan. Det går även att söka efter titel och ingredienser.

6 Slutsatser

Här görs en utvärdering av sitt arbete. Vad har gått bra och vad har gått dåligt? Vad hade kunnat göras annorlunda? Här kan man lägga in personliga åsikter om det egna arbetet.

Sidan generellt sätt fungerar bra. Den har några mindre buggar som skulle kunna redas ut men på grund av tidsbrist hinner det inte lösas. Buggarna är framförallt relaterade till sökfunktionerna.

Varje gång en användare gör en ändring i text input som är relaterad till sökning kommer en requests att skickas. Om användarens skriver väldigt snabbt eller håller in backspace kommer kommer användaren redigera texten snabbare än en response kommer tillbaka. Resultatet blir en hackande cursor. För att lösa detta borde antingen en request avbrytas om användaren trycker på en tangent med exempelvis en AbortController.

Ingredienserna som dom är just nu tillåter mellanrum i sig vilket är krav då ingredienser kan vara flera ord. Detta funkar för närvarande inte så bra med sökfunktionen då ingredienser antas vara ett ord. Detta upptäcktes sent i bugg testningen men sök funktionen är mer en bonus så dömdes vara tillräckligt bra.

React sidans kod är inte heller den bästa. Vissa sidor använder sig utav globala variabler som borde skrivas om till React refs. React refs va en React feature som också upptäcktes senare och eftersom det inte påverkar kodens funktionalitet skrevs inte de tidigare komponenterna om.

Det hade även varit önskvärt att ha en inloggningsfunktion och att recepten kopplats till användare. Som det är nu kan vem som helst redigera och ta bort recept som en annan har skapat vilket egentligen inte är önskvärt. På grund av detta har sidan fått sitt namn.

Källförteckning

[1] MDN ”Express/Node introduction” https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction hämtad 2022-01-16.

[2] MongoDB ”What is NoSQL” https://www.mongodb.com/nosql-explained hämtad 2022-01-16.

[3] MongoDB ”MongoDB & Mongoose: Compatibility and Comparison” https://www.mongodb.com/developer/article/mongoose-versus-nodejs-driver/#what-is-mongoose- hämtad 2022-01-16.

[4] mongoose-paginate-v2 ”mongoose-paginate-v2” https://github.com/aravindnc/mongoose-paginate-v2 hämtad 2022-01-16.

[5] Facebook ”A JavaScript library for building user interfaces” https://reactjs.org/ hämtad 2022-01-16.

[6] React Router ”Introduction” https://reactrouter.com/docs/en/v6/getting-started/tutorial hämtad 2022-01-16.

(1) siteFlow.drawio.png