CHALMERS |  GÖTEBORGS UNIVERSITET

MatCoachen

Utveckling av en motor för matplanering

Ett kandidatarbete inom Datateknik, Datavetenskap och Informationsteknik

Olof Berg Marklund
Emma Fahlen
Stefan Fritzon
Fredrik Kindström
Louise Rost
Lisa Snäll

Institutionen för Data- och informationsteknik
Handledare Fredrik Lindblad
Chalmers tekniska högskola
Göteborgs universitet

Göteborg, Sverige 31 maj 2017

FoodCoach
A software system for optimization of recipes and menu generation with the help of algorithms

OLOF BERG MARKLUND
EMMA FAHLÉN
STEFAN FRITZON
FREDRIK KINDSTRÖM
LOUISE ROST
LISA SNÄLL

Examiner: Niklas Broberg, Richard Johansson
Supervisor: Fredrik Lindblad

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden May 2017

Förord

Det här projektet gjordes som ett kandidatprojekt inom institutionen Data- och informationsteknik våren 2017. Projektet är ett samarbete mellan studenter på Chalmers tekniska högskola och Göteborgs universitet. Projektet genomfördes under en termin och innefattar 15 högskolepoäng.

Projektgruppen vill tacka Fredrik Lindblad för hans stöd och vägledning genom projektets gång. Projektgruppen vill också tacka examinator Richard Johansson och Niklas Broberg.

Idén att göra en matcoach kom när Emma och Fredrik vecka efter vecka la flera timmar på att planera veckomenyer som skulle vara miljövänliga, hälsosamma och billiga. Efter ett par år med denna tidskrävande göra så utbrast Fredrik: ”Men det här måste ju en dator kunna göra, vi gör ju samma sak vecka efter vecka! Det måste gå att automatisera”. I den stunden bestämde de sig för att de själva skulle skapa en automatisk smart hjälpreda i form av en webbapplikation. De skickade in ett kandidatarbetesförslag, fick det godkänt och den här rapporten är en beskrivning av resultatet från arbetet som följde.

MatCoachen: matplaneringsoptimering med hjälp av algoritmer

Ett mjukvarusystem för optimering av matrecept och menyer

OLOF BERG MARKLUND
EMMA FAHLÉN
STEFAN FRITZON
FREDRIK KINDSTRÖM
LOUISE ROST
LISA SNÄLL

Kandidatarbete vid institutionen för Data- och Informationsteknik, Chalmers tekniska högskola och Göteborgs universitet

Sammandrag

Denna rapport beskriver tillverkningen av en motor som ska ligga till grund för en framtida webb- och mobilapplikation som optimerar recept och genererar menyer. Systemet läser in existerande recept från recepthemsidor genom en parsningsprocess, och kan därefter optimera mängden av ingredienserna för att uppfylla användares näringsbehov, samtidigt som klimatavtrycket för receptet minimeras. Receptoptimeringen ger en personlig rekommendation utefter användarens profilinställningar. Motorn kan även rekommendera en meny med ett önskat antal recept med hänsyn till dessa inställningar samt med avseende på klimatavtryck, näringsintag eller rester. Parsning av recepthemsidor med hjälp av endast strängmanipulering som tillvägagångssätt uppvisade goda resultat. I ett test där 200 recept optimerades minskade klimatavtrycket med 9,6 % jämfört med det ursprungliga värdet. Systemet kan generera menyer med syftet att minimera klimatavtryck och optimera näringsintag, men gör det med hjälp av en algoritm med hög tidskomplexitet. Programmet är skrivet i programmeringsspråket Java med ramverket Play.

Nyckelord: Algoritmer, optimering, parsning, databaser

Abstract

This report presents the development of an engine which is to form the basis of a possible web and mobile application that can optimize recipes and generate menus. The system reads and gathers existing recipes from a recipe webpage by a parsing process, and can subsequently optimize the amount of the ingredients to make the recipe meet the users nutritonal needs, while the carbon footprint is minimized. The recipe optimization gives a personal recommendation according to the users profile settings. The engine can also recommend a menu consisting of a desired number of recipes while considering these settings, with respect to carbon footprint, nutrition intake or leftovers. The parsing of recipe webpages using only string manipulation as approach method showed good results. In a test where 200 recipes were optimized, the carbon footprint decreased by 9.6% compared to the original value. The system can generate menus with the aim of minimizing carbon footprint and optimizing nutritional intake, but do so by means of an algorithm of high time complexity. The program is written in the programming language Java with the Play framework.

Keywords: Algorithms, optimization, parse, databases

Ordlista

API - Application Programmable Interface, kommunikationen mellan en applikation och utomstående programvara

Backend - logikberäkningar och datahantering, t.ex. servern

BMR - Basic Metabolic Rate, mått på en människas energiförbrukning i vilande läge

Frontend - användargränssnittet, det som användaren ser och interagerar med, kommunikationsplattformen mellan användare och dator

HTTP - Hyper Text Transfer Protocol, kommunikationsprotokoll som används för att överföra webbsidor

JSON - JavaScript Object Notation, standardformat för att representera data

Målfunktion - Det uttryck som ska maximeras eller minimeras i ett optimeringsproblem

Parsning - analysering och tolkning av delar i en text

RDI - Rekommenderat Dagligt Intag, mängden vitaminer och mineraler en människa bör få i sig under ett dygn

Innehåll

1 Inledning

I Sverige kommer omkring 25% av klimatpåverkan från den mat vi äter och slänger [1]. Därför finns det stora miljövinster i att planera måltider bättre. Samtidigt finns ett stort intresse för hälsosam mat, men de många olika teorierna om hälsa och olika dieter gör det svårt att veta vilken mat som är hälsosam [2]. För att få hjälp med matplanering är det vanligt att söka sig till olika webb- och mobilapplikationer. Ett antal existerande applikationer för matplanering [3, 4, 5, 6] har undersökts, och visade sig sakna dels närings- och miljöuträkningar för menyer och även personligt anpassade recept.

1.1 Syfte

Projektet ämnar att resultera i en motor till en framtida webb- och mobilapplikation som hjälper människor att planera hälsosamma och miljövänliga måltider i sin vardag.

Information för att handla hälsosamt och miljövänligt ska inte behöva letas upp på egen hand av framtida användare; därför skapas en parsningsfunktion som gör det möjligt att hämta recept från webbsidor och för att få närings- och miljöinformation om ingredienserna i receptet. Algoritmer tas fram för att optimera ingrediensmängder i recept, det vill säga ändra proportionerna av ingredienserna, efter miljöpåverkan och användarens näringsbehov, samt för att jämföra olika recept och rekommendera de recept som bäst uppfyller användarens preferenser och behov i form av en meny. Projektet undersöker om tillgänglig data tillsammans med programkod kan konstruera realistiska och personligt optimerade maträtter och menyer.

1.2 Problembeskrivning

I detta avsnitt beskrivs problemen och uppgifterna mer utförligt. Funktionalitet samt systemkomponenter definieras, och avgränsningar presenteras.

1.2.1 Problemdefinition och mål

Projektets mål är att definiera vilka funktioner och komponenter som motorn ska stödja samt att implementera och skapa dessa för att sedan analysera resultatet.

Eftersom projektet ämnar att skapa en motor till en framtida applikation läggs inte fokus på att skapa ett avancerat användargränssnitt. Ett gränssnitt skapas endast med syftet att kunna demonstrera motorns funktionalitet. Tillagningsinstruktioner till recept tas inte hänsyn till utan endast ingredienslistor analyseras.

Projektet vilar på två grundpelare. Den första består av hämtning och tolkning av recept och officiella livsmedelsdatabaser och den andra består av algoritmer som genererar optimala recept och menyer för användare. Den första behövs för att det ska finnas välsorterad data i systemets databas som sedan kan användas under receptoptimeringen och menygenereringen. Projektet innefattar även ett användarprofilsystem där bland annat näringsinformation samt ålder, längd vikt och kön finns sparad om användare. Användarinformationen används av algoritmerna som optimerar recept och genererar menyer.

Hämtning och tolkning av recept och livsmedelsdatabaser
Den första grundpelaren kräver en process för hämtning av ingredienslistor tillhörande recept från en receptwebbsida för att sedan matcha ingredienserna mot livsmedel som finns i diverse livsmedelsdatabaser. Genom att implementera denna process tillhandahålls näringsinformation från recept hämtade på nätet som innan inte innehöll sådan information. Det krävs då en bra *parsningsfunktion* som klarar av att analysera och identifiera olika delar i en ingredienslista, vilken skapas med framförallt reguljära uttryck och strängmanipulering. En fråga som rapporten strävar efter att svara på lyder; hur goda resultat kan strängmanipulering och reguljära uttryck ge med uppgiften att identifiera rätt ingrediens och mängd i godtyckliga recept från nätet? Nedan presenteras funktionalitet och systemkomponenter som ska skapas.

Funktionalitet och systemkomponenter:

- Livsmedelsdatabas med information om livsmedel
- Hämtning av recept från internet
- Tolkning av hämtade recept och koppling av ingredienserna till livsmedelsdatabasen
- Receptdatabas med hämtade recept

Algoritmer
I algoritmdelen implementeras följande funktionaliteter:

- Optimering av ett givet recept med hänsyn till klimatavtryck och en användares näringsbehov
- Generering av meny med avseende på näringsintag
- Generering av meny med avseende på klimatavtryck
- Generering av meny med maximalt utnyttjande av givna ingredienser

För att skapa dessa funktionaliteter tas algoritmer fram. Optimering av ett recept innebär en omfördelning av proportionerna av ingredienserna i receptet på ett sådant sätt att klimatavtryck och näringsbehov blir så bra som möjligt. Klimatavtrycket är ett mått på hur mycket växthusgaser som släppts ut vid framtagningen av ett livsmedel, uttryckt i koldioxidekvivalenter per kilogram (CO_2e/kg) [1]. Klimatavtrycket ska tas hänsyn till vid både receptoptimering och menygenerering. Till menyerna tas en tillhörande inköpslista fram. Denna del av projektet har ingen specifik frågeställning, utan är snarare ett experiment med tillhörande analys. Rapporten ska analysera hur goda resultat receptoptimeringen kan ge med avseende på hälsa och miljö, samt undersöka olika metoder för att implementera en menygenerering med både gott resultat och god tidskomplexitet.

Hälsa och miljö är komplexa parametrar som kräver analys och mänskliga värderingar för att kunna implementeras till fullo. För att testa algoritmerna används därför endast rekommenderade näringsvärden, *RDI*, för representation av parametern hälsa, och klimatavtryck för representation av parametern miljö. Det recept som är närmast de rekommenderade näringsvärdena anses vara mest hälsosamt, och det recept som ger lägst klimatavtryck anses vara mest miljövänligt.

2 Teoretiskt underlag och definitioner

Detta kapitel introducerar teorier och tekniker som används i projektets slutprodukt. Den matematik, algoritmteori och näringslära som receptoptimeringen och menygenereringen grundar sig på beskrivs.

2.1 Linjärprogrammering

För att optimera recept utnyttjas linjärprogrammering [7]. Ett linjärt programmeringsproblem är ett optimeringsproblem med en linjär *målfunktion* och linjära bivillkor, exempelvis

$$\begin{aligned} &\textit{maximera } z = 5x_1 + 3x_2, \quad \text{(målfunktion)} \\ &\left.\begin{aligned} &2x_1 + x_2 \leq 4, \\ &3x_1 + 3x_2 \leq 5, \\ &x_1, x_2 \geq 0. \end{aligned}\right\} \text{(bivillkor)} \end{aligned}$$

Problemet är att hitta det maximala z inom området som bivillkoren definierar. För att göra det kan den så kallade simplexmetoden användas, en algoritm som söker på skärningspunkterna mellan linjerna, eller ytorna i fler dimensioner, som bivillkoren definierar.

För att problemet ska kunna skrivas om till likheter istället för olikheter införs två nya variabler, s_1 och s_2, så kallade slackvariabler. Problemet skrivs om enligt följande:

$$\begin{aligned} z - 5x_1 - 3x_2 &= 0, \\ 2x_1 + x_2 + s_1 &= 4, \\ 3x_1 + 3x_2 + s_2 &= 5, \\ x_1, x_2, s_1, s_2 &\geq 0. \end{aligned}$$

Ekvationerna matas in i en tabell enligt nedan.

z	x_1	x_2	s_1	s_2	HL
1	-5	-3	0	0	0
0	2	1	1	0	4
0	3	3	0	1	5

Eftersom det utöver målfunktionen finns två ekvationer med fyra variabler kan två av variablerna sättas till noll och de andra två lösas ut. Punkten ligger då på skärningen mellan två av de linjer som bivillkoren definierar. Genom att låta $x_1 = x_2 = 0$ fås att $s_1 = 4$ och $s_2 = 5$. Varje nollskild variabel ska ha en etta och resten nollor i sin kolumn i tabellen. Då har variabeln det värde som står på samma rad som ettan i dess kolumn, i kolumnen för högerledet (HL). Kravet uppfylls i det här fallet, i annat fall används elementära radoperationer för att uppfylla det.

Så länge det finns negativa värden i tabellens första rad går det att hitta en bättre

lösning, det vill säga $x_1 = x_2 = 0$ är inte optimalt eftersom det ger koefficienterna -5 och -3 i första raden. För att förbättra lösningen ska koefficienten för variabeln med den minsta koefficienten ökas så mycket det går utan att någon annan koefficient blir negativ. I det här fallet ska alltså koefficienten för x_1, det vill säga -5, ökas. För att se vilken variabel som begränsar x_1 löses x_1 ut ur de två nedre ekvationerna. Kom ihåg att $x_2 = 0$. Den första ekvationen ger $x_1 = 2 - 1/2s_1$, och den andra ger $x_1 = 5/3 - 5/3s_2$. Det innebär att x_1 maximeras med $s_1 = 0$. Nu finns återigen två nollvariabler (x_2 och s_1) och två nollskilda variabler (x_1 och s_2). Tabellen radreduceras med Gausseliminationen så att de nollskilda variablerna har en etta och resten nollor i sin kolumn. Sedan upprepas samma process tills dess att alla koefficienter på första raden är positiva. Då är den optimala lösningen funnen.

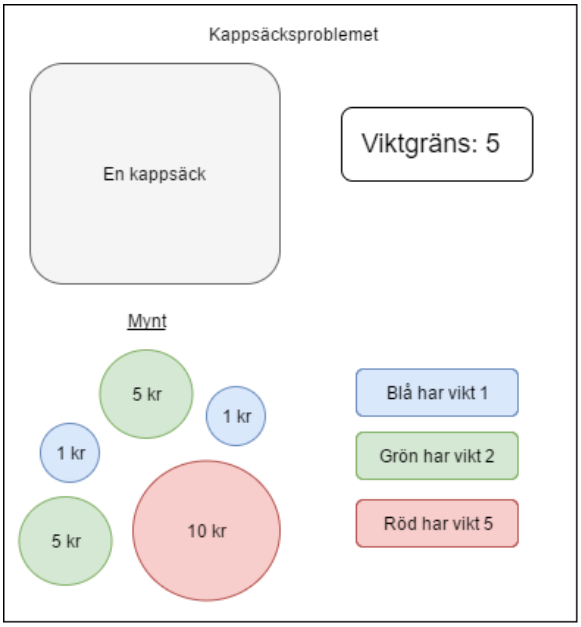
Simplexmetoden har polynomiell genomsnittlig tidskomplexitet [8].

2.2 Algoritmteori

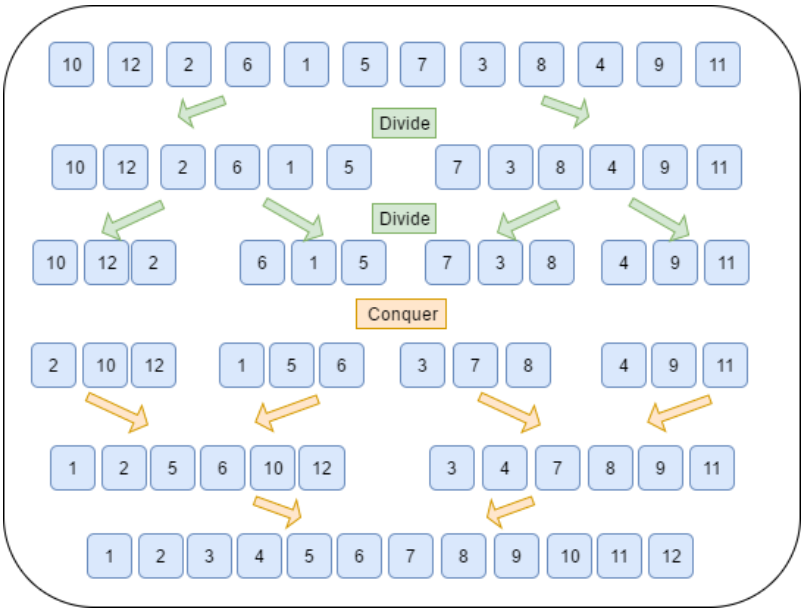
För att generera menyer används vedertagna algoritmteorier. Dynamiska programmeringsalgoritmer, divide and conquer-algoritmer och giriga algoritmer beskrivs i *Algorithm Design* [9].

Enligt *Algorithm Design* [9] är en girig algoritm en algoritm som inte sparar information i varje steg, utan endast utgår från informationen i nuvarande punkt för att räkna på nästkommande steg. Exempelvis kan en girig algoritm användas i det så kallade kappsäcksproblemet, där det mest värdefulla elementet som får plats i kappsäcken väljs ut och placeras i kappsäcken upp till en given viktgräns. Varje element har en vikt och ett värde. I figur 1 finns ett exempel på kappsäcksproblemet, där viktgränsen är fem viktenheter och det finns tre olika sorters element; blå med 1 i värde och 1 i vikt, grön med 5 i värde och 2 i vikt och röd med 10 i värde och 5 i vikt. Med den giriga algoritmen väljs alltså det bästa elementet som får plats i kappsäcken ut och läggs ner. I detta exempel väljs det röda myntet med värdet 10, men två gröna mynt och ett blått mynt ger ett värde på 11 enheter, vilket är ett bättre värde. En girig algoritm ger alltså inte alltid ett optimalt resultat.

I divide and conquer delas problemet upp i mindre delar för att sedan lösa delproblemen var för sig. Ett vanligt exempel på divide and conquer är mergesort där problemet delas upp i mindre bitar, vilket minskar omfattningen av varje del. Resultaten av dessa delar läggs sedan ihop till ett gemensamt resultat, vilket visas i figur 2.



Figur 1: Ett bildexempel på ett kappsäcksproblem där målet är att maximera värdet av mynten i kappsäcken utan att överstiga viktgränsen.



Figur 2: Ett bildexempel på en divide and conquer-algoritm.

Enligt *Algorithm Design* [9] utnyttjas styrkorna från både giriga algoritmer och divide and conquer-algoritmer i dynamisk programmering, där delproblemen räknas ut på ett effektivt sätt. Sedan kan resultaten användas för att underlätta beräkningen av andra delproblem. Det sker genom rekursion samt en maximering eller minimering av uträkningarna för att få tag på den bästa kombinationen av en given mängd element. Formel 1 nedan kan användas för att utföra dynamisk programmering. *F(n,V)* är optimeringsfunktionen. I varje steg väljs ett av n antal element ut för att läggas till i det totala värdet V.

$$F(n, V) = min/max\left(F(n-1, V+i), F(n-1, V)\right). \tag{1}$$

2.3 L_2-normen

För att generera en meny med optimalt näringsintag kommer L_2-normen [10] att användas. Normer används inom matematiken för att sätta ett värde på avståndet från ett objekt till ett annat. Här används L_2-normen för att ge ett värde på avståndet mellan en aktuell vektor och en optimal vektor. Den aktuella vektorn betecknas $\mathbf{x}$. L_2-normen är då definierad som roten ur summan av absolutbeloppen av elementen i kvadrat, det vill säga

$$|\mathbf{x}| = \sqrt{\sum_{j=1}^{n} |x_j|^2}, \tag{2}$$

där n är antalet element i $\mathbf{x}$. I det tvådimensionella fallet blir L_2-normen alltså

$$|\mathbf{x}| = \sqrt{\sum_{j=1}^{2} |x_j|^2} = \sqrt{x_1^2 + x_2^2},$$

även känt som Pythagoras sats.

2.4 Näringsrekommendation

För att optimera en algoritm utifrån parametern hälsa måste begreppet definieras. Det pågår ständigt debatter, studier och upptäckter inom näringslära och välmående [2]. Projektets definition av den optimala dieten med avseende på hälsa är att dieten ska uppfylla RDI av livsnödvändiga näringsämnen i högsta möjliga mån, samt ligga under mängden för överdos. När det gäller referensvärden för intag av näringsämnen i form av vitaminer och mineraler följer projektet NNR 2012 (Nordiska näringsrekommendationer 2012) [11].

Harris och Benedict [12] har i studien *A biometric study of basal metabolism in man* definierat en ekvation som går ut på att att räkna ut Basic Metabolic Rate (*BMR*), det vill säga en persons kaloriförbrukning i vilande läge. Resultatet av ekvationen beror på kön, vikt, längd, och ålder. En utökad version av denna ekvation används i detta projekt för att räkna ut kaloribehov. För att få en mer realistisk bild av en persons kaloribehov utgår ekvationen från en användares fysiska aktivitetsnivå samt om användaren vill bibehålla, öka eller minska i vikt. Den fysiska aktivitetsnivån för användaren estimeras utifrån tabell 1: 'Fysisk aktivtetsnivå', se s. 12. Den eventuellt önskade viktändringen

lägger till eller tar bort ett visst antal kalorier på det uträknade kaloribehovet.

De energigivande näringsämnena är fett, kolhydrater och protein. 1 g fett ger 9 kcal medan 1 g kolhydrater respektive protein ger 4 kcal [13]. Fördelning mellan energigivande näringsämnen följer NNR 2012 och är rekommenderat till 10-20 E% protein [11, s. 281–310], 40-60 E% kolhydrater [11, s. 249–280] och 25-40 E% fett [11, s. 217–248]. E% står för energiprocent, det vill säga hur många procent av energin som kommer från respektive näringsämne.

I Livsmedelsverkets rapport från 2014 beskrivs referensvärdena på följande sätt: ”I NNR 2012 har referensvärden för intag av näringsämnen satts utifrån vetenskapligt underlag om vilka intag som garanterar optimal nutrition och bidrar till att förebygga kroniska sjukdomar.” [14, s. 3].

3 Teknisk bas

I detta avsnitt beskrivs de viktigaste och mest betydelsefulla program, bibliotek och verktyg som användes under projektet.

3.1 Play 2

Play framework 2 är ett ramverk som möjliggör utvecklandet av webbapplikationer i Java [15]. Play innehåller funktionalitet för att sätta upp en server, svara på *HTTP*-anrop och prata med en databas.

3.2 MySQL

SQL står för Structured Query Language [16] och är ett vida spritt språk för att prata med relationsdatabaser. I dessa databaser delas datan upp i tabeller vars rader motsvarar en datapunkt och varje kolumn ett attribut. Kolumnerna kan dessutom länkas till andra tabeller för att bygga upp relationer mellan olika datapunkter. MySQL är ett databassystem som använder sig av SQL som språk [17].

3.3 Ebean

Datarepresentationen hos relationella databassystem som MySQL och objektorienterade programmeringsspråk som Java skiljer sig åt en hel del. Det finns ingen självklar formel för att översätta ett Java-objekt med instansvariabler till en SQL-tabell med kolumner. Traditionellt sett skrivs det kod för att konvertera mellan dessa två format, för att spara ett visst objekt, för att hämta vissa objekt med mera. Denna kod kan bli väldigt repetitiv och tidskrävande att skriva eftersom det ofta innebär mycket upprepning i kodmönstret. För att underlätta detta moment finns Ebean.

Ebean är en Object Relational Mapper [18]. Ett sådant bibliotek översätter, med hjälp av annoteringar, ett Java-objekt till sin databasrepresentation automatiskt. Istället för att skriva ren SQL för allt som händer genereras SQL efter ett mer objektorienterat gränssnitt.

3.4 JSON Tagger

JSON Tagger [19] är ett API som analyserar svenska texter och ger grammatisk information om orden i texten. Detta API avgör vilka ord som är adjektiv, substantiv, vilken form de är skrivna på med mera. JSON Tagger baseras på UDPipe [20] vilket är ett mångsidigt programpaket för uppträning av maskininlärningsmodeller för igenkänning av text.

3.5 Övriga javabibliotek

Play-servern använder sig även av följande bibliotek:

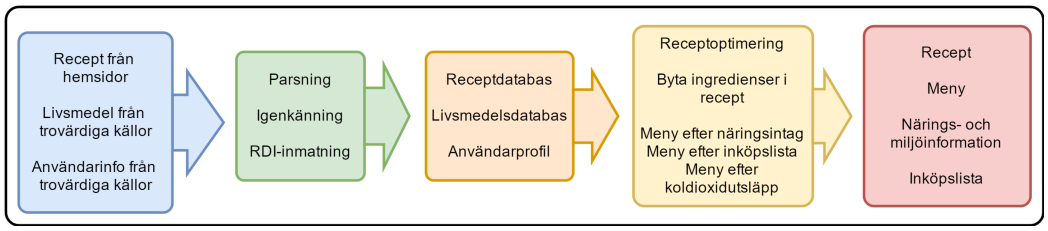
- **Apache Commons Math** [21] är ett Java-bibliotek som tillhandahåller matematiska funktioner. Den metod som användes i detta projekt är en funktion som räknar ut levenshteinavståndet mellan två strängar. Med levenshteinavstånd menas hur många ändringar som krävs för att två strängar ska vara identiska, det vill säga ett mått på hur lika varandra de är. Projektet använde även bibliotekets implementation av simplexmetoden.
- **Crawler4j** [22] är ett hjälpbibliotek som skapar flera parallella trådar som var och en hämtar data från en webbadress. Detta tillvägagångssätt snabbar upp processen jämfört med seriella metoder.
- Parserbiblioteket **jSoup** [23] används för att extrahera text ur HTML-element som finns i webbsidor. HTML är i sig bara text men biblioteket kan navigera sig fram i den nästlade strukturen och ta ut innehåll samtidigt som det skalar bort programmatiska nyckelord.

3.6 Webbibliotek och ramverk

Webbsidan som utvecklades använde följande ramverk och bibliotek:

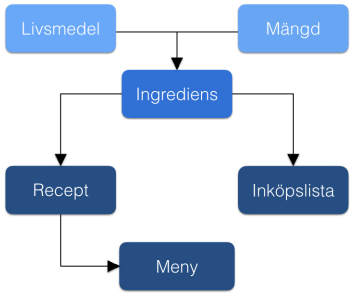
- **Vue.js** [24] är ett ramverk skrivet i programmeringspråket javascript. Ramverket hanterar bland annat asynkrona uppdateringar av webbsidekomponenter baserat på datauppdateringar.
- JavaScript-biblioteket **JQuery** [25] innehåller många funktioner för animationer, händelsehantering och underlättar kommunikationen mellan *frontend* och *backend*.
- **Bootstrap** [26] är ett responsivt ramverk för HTML, CSS och JavaScript som underlättar vid designfasen och gör gränssnittet adaptivt till både datorer, mobiler och läsplattor.

4 Metod



Figur 3: Flöde av information genom motorn, från vänster till höger omvandlas data till olika former för att resultera i vad som finns i den röda rutan.

MatCoachen består av ett antal huvudkomponenter. I botten ligger en livsmedelsdatabas med tillhörande datamodeller samt en användarmodell som övriga moduler i sin tur bygger på, se figur 3 ovan. Tolkningsmodulen ansvarar för att omvandla rådata i form av recept på nätet till receptobjekt som är matchade med livsmedel från livsmedelsdatabasen. Efter denna process används resulterande data, tillsammans med data från användarprofilen, för att utföra beräkningar för receptoptimeringen och menygenereringen. Systemet vilar på de grundläggande datamodellerna i figur 4 nedan, vilka återkommer vid flera av projektets abstraktionsnivåer.

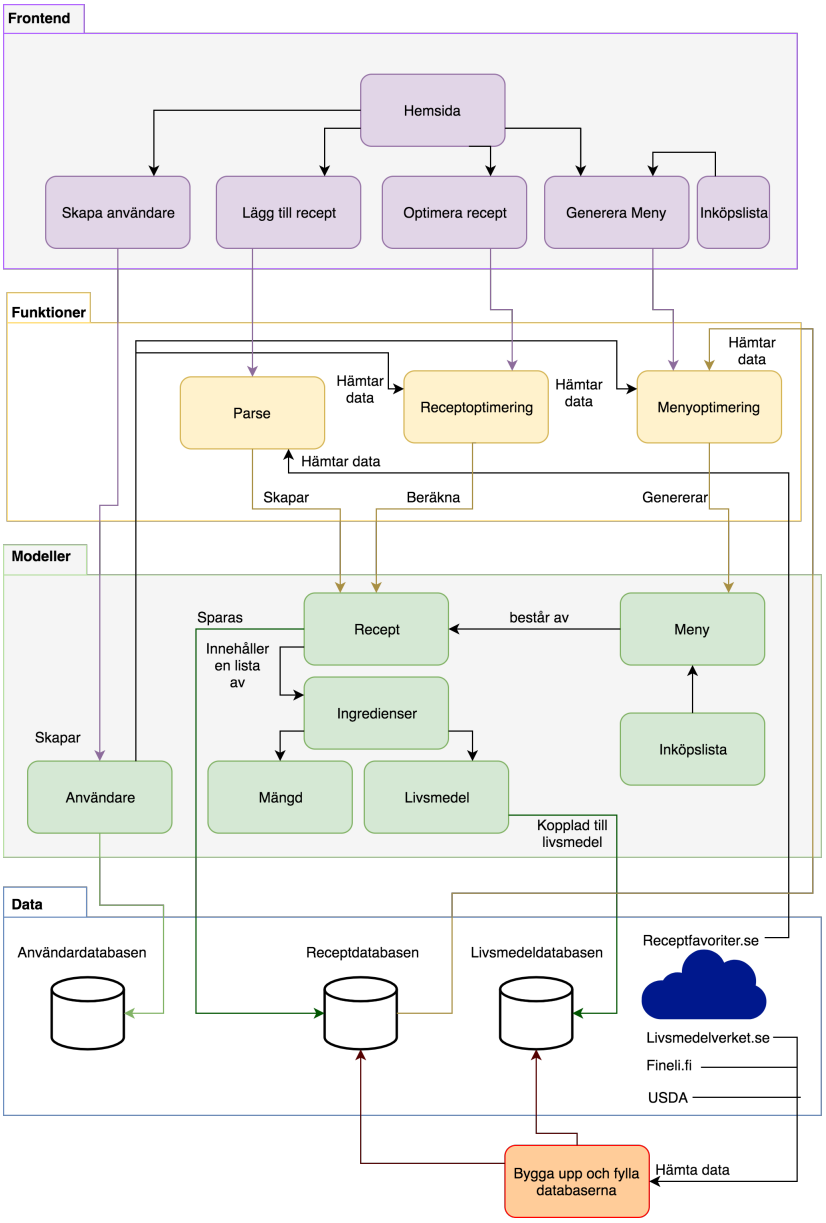


Figur 4: Hur systemets grundläggande datamodeller förhåller sig till varandra.

Livsmedel i systemet är modellerade med två klasser: livsmedel och livsmedelsgrupp. Livsmedelsgruppen representerar en större samling som i grunden är samma livsmedel men som till exempel är behandlade på olika sätt eller innehåller olika tillsatser. I livsmedelsklassen finns näringsdatan som kan komma från olika källor. Se avsnitt 4.2 för mer om det.

En ingrediens innehåller ett livsmedel och en mängd av livsmedlet. Mängd är en egen klass som har information om hur mycket av livsmedlet som ingrediensen består av samt i vilken enhet det anges. Se avsnitt 4.2 för mer om det.

Ett recept utgörs av en titel, en lista med ingredienser samt antal portioner receptet är avsett för. En lista av recept bygger i sin tur upp en meny, vilken är till för att underlätta uträkningar vid menygenereringen. Vid menygenereringen skapas även en tillhörande inköpslista vilken utgörs av ingredienser.



Figur 5: Övergripande bild över motorns alla komponenter och hur de samverkar.

4.1 Användarprofil

Användarprofilen är till för att spara den data som är relevant för att optimera en kosthållning för en individ. Denna data sparas ned i en databas och används när algoritmerna ska optimera avnseende på hälsa. Datan som lagras i användarprofilen kan delas in i tre kategorier som presenteras nedan. Det följer även beskrivningar av hur datan tas fram genom beräkningar.

- **Rekommenderat kaloribehov och energigivande näringsämnesfördelning**
  Indata som krävs för denna beräkning är ålder, kön, vikt, längd, fysisk aktivitet och om användaren vill bibehålla, minska eller öka i vikt. Ekvationen kommer från Harris & Benedict-ekvationen 1918 [12]. I första steget beräknas användarens kaloriförbukning i viloläge, alltså BMR.

  Harris & Benedict-ekvationen för kvinnor:

  $$\textbf{BMR} = 655,0955 + (9,5634 \cdot \text{kroppsvikt}) + (1,8496 \cdot \text{längd}) - (4,6756 \cdot \text{ålder}).$$

  Kroppsvikt anges i kilogram, längd i centimeter och ålder i år. Med samma enheter ges Harris & Benedict-ekvationen för män av följande ekvation:

  $$\textbf{BMR} = 66,5 + (13,7516 \cdot \text{kroppsvikt}) + (5,0033 \cdot \text{längd}) - (6,7550 \cdot \text{ålder}).$$

  I steg två multipliceras användarens BMR med ett tal som motsvarar användarens fysiska aktivitetsnivå. Användaren estimerar sin aktivitetsnivå utifrån följande tabell.

  Tabell 1: Fysisk aktivitetsnivå

Fysisk aktivitetsnivå	**Dagligt kaloribehov**
Liten till ingen motion	1,2 · **BMR**
Lätt motion(1–3 dagar i veckan)	1,35 · **BMR**
Måttlig motion(3–5 dagar i veckan)	1,55 · **BMR**
Tung motion(6–7 dagar i veckan)	1,725 · **BMR**
Väldigt tung motion (två gånger om dagen, extra tung)	1,9 · **BMR**

  Om användaren har angett en önskad viktändring kommer kaloribehovet minska eller öka med 500 kalorier.

  I projektet räknas med en energifördelning på 30 E% från fett, 55 E% från kolhydrater och 15 E% från protein. Värdena är tagna från livsmedelsverkets rapport Bra livsmedelsval baserat på nordiska näringsrekommendationer 2012 [14]. Genom att multiplicera dessa värden med energibehovet för användaren och sedan dividera med 9 för fett respektive 4 för kolhydrater och protein, fås behovet av respektive näringsämne angett i gram.

- **Vitamin- och mineralrekommendationer**

  Rekommendationer för vitaminer och mineraler baseras endast på ålder och kön. Datan hämtas från tabellerna i bilagorna från Livsmedelsverkets rapport Bra livsmedelsval baserat på nordiska näringsrekommendationer 2012 [14]. Även data för överdosering finns tillgänglig i användarprofilen. Denna data är i första hand hämtad från Livsmedelsverket [27], men kompletteras av värden hämtade från Kureras hemsida [28] eftersom Livsmedelsverket saknar vissa värden. Dessa värden läggs alltså in manuellt i användarprofilen.

- **Matpreferenser**

  Här samlas data om vilka ingredienser användaren vill undvika för att vid menygenereringen kunna utesluta recept som innehåller dessa ingredienser.

Målet med denna modell, utöver att ligga till grund för uträkningarna till de personligt optimerade recepten och genererade menyerna, är att vara omfattande nog för framtida ändamål. Visionen är att användarprofilerna ska implementeras genom en webbapplikation och sparas i en databas. Databasen är därför skapad med utrymme för att spara annan information som kan behövas vid framtida funktioner. Därför finns det kolumner för bland annat email, födelsedag och tidpunkt då användaren blev registrerad på sidan.

För att underlätta testerna av de olika optimeringsfunktionerna implementerades en så kallad standardanvändare vars värden är satta efter generella näringsrekommendationer för en person från Norden.

4.2 Representation av livsmedel och recept

Livsmedelsdatabasen består av livsmedelsobjekt med tillhörande näringsinformation. Dessa behöver vara organiserade på ett sätt som underlättar parsningssprocessen samtidigt som näringsinformation för beräkningarna smidigt kan tillhandahållas. Utöver livsmedel finns datamodeller för att representera ett recept samt enstaka ingredienser i ett recept. I detta avsnitt beskrivs dessa modeller mer i detalj.

4.2.1 Informationskälla för livsmedel

Livsmedelsverket upprätthåller en databas med analyserade livsmedel som finns på den svenska marknaden. För vart och ett av dessa livsmedel återfinns analyserad data på över 50 näringsämnen [29]. Livsmedlena är dessutom kategoriserade enligt två europeiska standarder för klassificering av livsmedel, EuroFIR [30] och LanguaL [31]. Klassificeringarna är ytterst utförliga och innehåller mängder av delkategorier för varje liten del av varifrån livsmedlet härstammar.

Finlands motsvarighet till Sveriges livsmedelsdatabas heter Fineli och hanteras av Institutet för hälsa och välfärd i Finland [32]. Fineli är av samma karaktär som Livsmedelsverkets databas, och innehåller både livsmedel och färdiga maträtter som finns på marknaden i Finland. Den följer inte samma kategoriseringssystem utan har ett eget. Här finns också information om vilka dieter och allergier varje livsmedel är lämpat för, det vill säga om livsmedlet innehåller exempelvis gluten, laktos, kött eller ägg.

U.S. Department of Agriculture i USA har även de en databas [33]. Databasen innehåller överlägset flest livsmedel och tillhandahåller dessutom olika mått för varje livsmedel, inte bara näringsvärden per 100 gram. Projektet ämnade först att basera datamodellerna på Livsmedelsverkets livsmedelsdatabas men övergick senare till att i första hand använda Fineli med kompletteringar från Livsmedelsverket. Anledningen är att Fineli har bäst struktur på livsmedelsnamnen, den är helt enkelt bättre organiserad än Livsmedelsverkets databas. Mot slutet adderades även stöd för amerikanska USDAs databas.

De tre databaserna går i sin enklaste form att representera som ett kalkylark. Nedan syns ett exempel på ett antal rader hämtade från Fineli. Varje rad representerar ett enskilt livsmedel med ett unikt nummer, namn samt näringsinformation för flera olika näringsämnen.

Tabell 2: Utdrag ur kalkylark hämtat från Fineli

id	name	energi (kJ)	kolhydrater (g)	fett (g)	protein (g)	...
34694	Papaya, skalad	164	7,8	0,3	0,5	...
33060	Papaya, torkad	1129	57,6	0,9	4	...
387	Paprika, grön	89	2,5	0,3	0,9	...
388	Paprika, gul	126	5,1	0,2	0,9	...

4.2.2 Livsmedel och livsmedelsgrupper

Under implementationsprocessen togs beslutet att tilldela varje rad i rådatan en egen grupp som klumpar ihop livsmedel som har med varandra att göra. Denna struktur behövs eftersom parsningssprocessen vilar på den (se avsnitt 4.3), samt eftersom det underlättar administreringen av databasen. En livsmedelsgrupp är alltså en grupp med livsmedel som kan betraktas som samma men förekommer med olika variationer. I exemplet ovan blir paprika en grupp och papaya en annan.

Tabell 3: Livsmedelsgrupper för Fineli

id	group-name	default	tags	display-name	extra-tags
34694	Papaya	TRUE	skalad	Papaya	papayor, papayafrukt
33060	Papaya		torkad	Torkad papaya	
387	Paprika		grön	Grön paprika	
388	Paprika	TRUE	gul	Gul paprika	

Kolumnen 'default' indikerar ifall livsmedlet är standardlivsmedel för den aktuella livsmedelsgruppen. 'Skalad papaya' är alltså standard för livsmedelsgruppen 'Papaya'. Detta system används också i parsningsprocessen. Det möjliggör dessutom en generaliserad struktur som kan anpassas till alla tre källorna (svenska, finska och amerikanska). Arket ovan (tabell 3) har samma struktur för alla källor. Med hjälp av id-numret på livsmedlet kan näringsinformationen matas in i databasen genom att titta i varje databas respektive rådataark (tabell 2). Grupperna är alltså fristående från vilken källa varje livsmedel

kommer från.

Näringsvärden för varje livsmedel anges per 100 gram. Utöver söktaggar, visningsnamn och näringsdata finns även livsmedelsklassifikation av varje livsmedel. Detta system är hämtat från Fineli. Exempelvis har 'torkad basilika' klassificeringen 'torkade örter', medan 'bondbönor' hör till 'baljväxter'.

4.2.3 Recept och ingredienser

Ett recept består av en lista av ingredienser, en titel samt en indikation på hur många portioner receptet är avsett för. En ingrediens består av ett livsmedel och en mängd. Mängd är i sin tur en egen klass. Denna klass innehåller uppräkningstyper för olika enheter med tillhörande kvoter för omvandling mellan dessa. Nedan följer ett utdrag ur mängdklassen Amount.java.

```
// Standard SI-Volume
LITER(10, Type.VOLUME, new String[] { "l", "liter" }),
DECILITER(1, Type.VOLUME, new String[] { "dl", "deciliter", "deci liter" }),
...
// Standard SI-Mass
KILOGRAM(10, Type.MASS, new String[] { "kg", "kilo", "kilogram", "kilo gram" }),
...
// Swedish cookbook units
KRYDDMATT(0.01, Type.VOLUME, new String[] { "krm", "kryddmatt" }),
TESKED(0.05, Type.VOLUME, new String[] { "tsk", "tesked" }),
...
// Single piece unit
STYCK(0.0, Type.SINGLE, new String[] { "st", "stycken" } ),
...
```

För att kunna beräkna korrekta värden för livsmedel som är angivna i volymsenheter i ett recept behövs en konstant som anger livsmedlets densitet. För ingredienser som är angivna på formen 'en banan', det vill säga med en angivelse om antal, krävs dessutom information om hur mycket ett exemplar av livsmedlet väger. Denna data har lagts till i databasen manuellt för varje livsmedel. Med hjälp av klassificeringen hos varje livsmedel beräknas även ett uppskattat värde på klimatavtryck för respektive ingrediens. Värdet består av data från Mat-klimat-listan [1] multiplicerat med mängden av ingrediensen. Värden för klimatavtrycket är även de tillagda manuellt för varje klassificering.

En ingrediens skalar alltså sitt livsmedels näringsvärde och övrig data med avseende på mängden. På samma sätt har ett recept logik för att addera ihop alla sina ingrediensers näringsdata. Dessa metoder används sedan vid receptoptimeringen och menygenereringen.

4.3 Insamling och tolkning av recept

I det här avsnittet beskrivs hur insamling av data går till och hur parsningsprocessen hanteras, vad för data som krävs för att parsningsprocessen ska vara genomförbar samt

hur denna data är strukturerad och kategoriserad för att uppnå ett så korrekt resultat som möjligt.

4.3.1 Extrahera information från webbplatser

På internet finns mängder av recept. Enligt projektets efterforskningar består en majoritet av dem endast av information i form av enkel text. Det finns ingen länk till vidare information om de ingredienser som inkluderas i ett recept. Istället används ofta taggar och manuellt inlagda kategorier för recept, exempelvis 'indisk mat' och 'lättlagat', eller kategorier för huvudingredienser, exempelvis 'kyckling' och 'grädde' [34]. Detta system är enkelt och fungerar i isolerade fall bra.

Ett problem är dock att det krävs mycket administration för att hålla dessa kategorier konsekventa och det finns ingen möjlighet att räkna på recepten. Med det menas att det inte finns möjlighet att räkna ut värden för näring eller jämföra ingredienser med ett liknande recept eftersom det inte går att unikt identifiera ingredienser utifrån endast textsträngar. '2 gula lökar' är inte likadan som '1 gul lök' till exempel. Datan som finns direkt tillgänglig är således väldigt smal och behöver formateras för att bli användbar för storskaliga beräkningar.

4.3.2 Rådata

I projektets startskede cirkulerade tankar på att analysera ett flertal receptsidor, bland annat för att kunna utföra analys på en större mängd data. Under projektets gång koncentrerades dock arbetet enbart på en godtycklig webbplats, nämligen receptfavoriter.se [35]. Under utvecklingen var det enklare att arbeta mot en enskild sida, valet föll då på receptfavoriter.se [35] eftersom kod implementerades för den först.

Projektet använder sig av crawler4j [22] som är ett öppet Java-bibliotek för att hämta information från en mängd webbadresser samtidigt. För informationshämtningen behövs alltså en lista på webbadresser att läsa av. Den erhålls genom receptfavoriters [35] sitemap. En sitemap är en lista på alla webbadresser hos en domän, i detta fall är det alltså en lista på alla recept på receptfavoriter.se [35]. Denna fil finns för att hjälpa sökmotorer att indexera sidan och kan således också användas för att indexera i andra syften.

För att ta ut enskilda textsträngar ur HTML används biblioteket jSoup [23] som läser råtext ur HTML-elementen. På detta sätt extraheras text från respektive recepts webbadress. Listan med ingrediensträngar skickas sedan vidare till tolkningsmodulen.

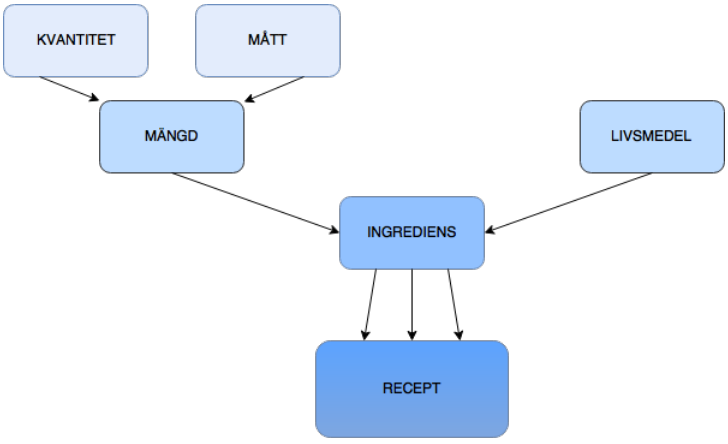
Figur 6: Exempel på textsträngar som extraheras från HTML. Källa: [34]

Textsträngarna som behöver identifieras kan vara av enkel karaktär som ’0,5 dl olivolja’ eller ha mer invecklad struktur som ’0,5 melon som honungs- eller cantaloupe i klyftor’, där den senare kräver ett betydligt mer komplext system för att kunna hanteras korrekt.

4.3.3 Tolkning av hämtad information

När en lista med strängar har hämtats från receptets webbadress ska dessa strängar tolkas så att det är möjligt att identifiera vad som är livsmedel, mått, mängder respektive kommentarer. I slutsteget av den processen ska ett receptobjekt skapas som består av flera ingrediensobjekt. Dessa ingrediensobjekt består i sin tur av ett livsmedel, en kommentar, en titel samt ett mängdobjekt. Mängdobjektet består i sin tur av ett mått och en kvantitet.

För att kunna skapa ett receptobjekt måste alltså alla dessa komponenter i grundsträngarna identifieras. Ingrediensträngarna i listan processeras en i taget tills hela listan är bearbetad.



Figur 7: Visualiering av hur komponenterna i ett recept hör samman.

4.3.4 Matchning mot livsmedelsdatabasen

Alla livsmedel som är standardlivsmedel för livsmedelsgrupper har tillhörande söksträngar. Om ingrediensträngen som bearbetas i parsningsprocessen innehåller någon av dessa söksträngar går det att anta att just den delen av ingrediensträngen är en ingrediens. Det är genom söksträngarna samt namnen på livsmedelsgrupperna som parsern hittar ingredienser i ingrediensträngarna.

Alla söksträngar är unika; skulle de inte vara det skulle parsern inte veta vilken av livsmedlen med samma söksträng som bör väljas. Söksträngen 'köttfärs' kan alltså inte vara med som söksträng för både livsmedelsgruppen 'nötfärs' och livsmedelsgruppen 'fläskfärs'. Söksträngarna för standardlivsmedlen i varje livsmedelsgrupp representerar samtliga livsmedel som finns i den kategorin så att även de livsmedlen kan identifieras. Dessa söksträngar är manuellt inlagda för varje livsmedelsgrupp.

Varje livsmedel har även sina egna söksträngar som ofta är adjektiv till sin tillhörande livsmedelsgrupp. Dessa finns för att livsmedlen ska kunna identifieras inom sin kategori. Söksträngarna behöver inte vara unika. Exempelvis har livsmedelsgruppen 'mango' 'färsk mango' som sitt standardlivsmedel, men i samma livsmedelsgrupp finns ett livsmedel 'torkad mango' som identifieras genom att den har sin egna söksträng 'torkad'. På det sättet hittar parsern det specifika livsmedlet 'torkad mango'.

4.3.5 Parsningsprocess

När en ingrediensträng hämtats startar parsningsprocessen. Denna process består av ett antal steg som är ordnade på ett sådant sätt att orden som strängen innehåller ska tolkas och mappas med ett så korrekt resultat som möjligt. Första steget är att titta på vad som kan antas från början inte är en ingrediens, mängd eller mått. Strängen undersöks efter text inom parentes. Finns det text i parentes plockas den bort ur originalsträngen och sparas som kommentar.

Nästa steg är att undersöka om det finns något kolon med i den resterande strängen, i så fall går det att anta att det som står innan kolonet är en titel på ett delrecept i originalreceptet. Det förmodade delreceptet sparas undan och tas sedan bort från strängen som arbetas med. Efter det undersöks om den resterande strängen innehåller 'eller'. Om den gör det läggs 'eller' och strängen efter det ordet till i kommentar och tas bort från arbetssträngen.

Nästa steg i processen är att hitta mått och kvantitet i texraden. Kvantitet hittas genom att anropa API:t Json Tagger [19]. API:t svarar och har bland annat taggat alla ord som är nummer. Detta nummer sätts som kvantitet och tas bort från strängen som arbetas med. Mått hittas genom att matcha resterande sträng mot alla listade mått. Om inget mått hittas sätts mått till 'styck'. Hela tiden tas identifierade ord bort från arbetssträngen. Mått och kvantitet blir nu ett mängdobjekt. Om varken mått eller kvantitet lyckas identifieras sätts mängden till noll vilket slutligen nollställer alla näringsvärden.

Ingrediensen identifieras genom att gå igenom livsmedelsdatabasens alla söksträngar och

se om någon av dessa matchar någon del av den resterande strängen. Först hittar parsern en livsmedelsgrupp, och sedan letar den efter ett livsmedel. Hittar den inget livsmedel använder den sig av standardlivsmedlet i den identifierade livsmedelsgruppen. När ingrediensen är identifierad tas den delen av strängen bort.

Den eventuella strängen som återstår plus tidigare borttagen text läggs som kommentar på ingrediensen. När alla ingrediensssträngar i ett recept har gått igenom ovanstående process och blivit ingrediensobjekt skapas ett receptobjekt. Kommentaren är tänkt att hjälpa användaren i ett senare skede, den kan innehålla saker som exempelvis 'att kokas' eller 'för garnering'. Färdiga recept sparas i applikationens receptdatabas.

4.3.6 Autocorrect

Om en livsmedelsgrupp inte lyckas identifieras testas teorin att ingrediensen i ingrediensssträngen kan ha en alternativ stavning till söksträngarna som finns i livsmedelsdatabasen. Det testas genom att den resterande strängen skickas in till metoden autocorrect som undersöker om ord i strängen är en viss distans från att matcha en sökträng i livsmedelsdatabasen. Om distansen mellan sökströngen och ordet ligger i ett förbestämt intervall antas det att ordet i ingrediensssträngen syftar till den livsmedelsgrupp som den aktuella sökströngen är kopplad till. Då är livsmedelsgruppen hittad och parsningsprocessen fortsätter till nästa steg; att försöka hitta ett livsmedel.

Distansen mellan två strängar beräknas med hjälp av levenshteinavståndet som biblioteket Apache Commons Math [21] tillhandahåller en funktion för. Den tillåtna maxdistansen som differensen mellan två strängar får ha för att räknas som samma livsmedelsgrupp sattes till 1. Att distansen är 1 innebär att endast en ändring i någon av strängarna behöver göras för att de ska bli identiska. Det ger en så liten felmarginal som möjligt. På det sättet kan exempelvis särskrivningar behandlas, men även apostrofer och felstavningar.

4.4 Receptoptimering

Vid receptoptimeringen är målet att finna optimala proportioner av ingredienserna för ett givet recept, anpassat efter användarens behov samtidigt som klimatavtrycket för receptet minimeras. För att göra det används simplexmetoden. Eftersom näringsbehovet från användarprofilen är angett per dag behöver dessa värden skalas om. I projektet antas att en måltid utgör 30% av det dagliga matintaget.

4.4.1 Hälsa och miljö

Värden för rekommenderat intag av energi, fett, protein och kolhydrater hämtas från användarprofilen. Miljöaspekten kommer in vid utförandet av optimeringen. Det kommer finnas en mängd olika varianter av receptet, det vill säga med olika proportioner av ingredienserna, som uppfyller kraven. Vilket alternativ som lämnas till användaren avgörs av hur stort klimatavtryck respektive alternativ ger. Det är alltså det alternativ med lägst totalt klimatavtryck som returneras och rekommenderas för användaren.

4.4.2 Optimeringsalgoritm

För att lösa linjäroptimeringsproblemet användes återigen Apache Commons Math [21]. Nedan följer en beskrivning av hur simplexmetoden används i projektet.

På axlarna i ett n-dimensionellt rum sätts mängden av respektive ingrediens, där n är antalet ingredienser i receptet. Gränser sätts för hur mycket av respektive ingrediens det optimerade receptet ska innehålla. Dessa gränser behövs för att hålla receptet någorlunda likt originalreceptet. För att förenkla används en och samma procentsats för samtliga ingredienser. Den undre begränsningen sätts fixt till 75% av originalvärdena, medan den övre från början sätts till 150%. Utöver det sätts även krav på att receptet ska ge 80% av rekommenderat intag av energi, fett, protein och kolhydrater för användaren, vilket ger undre begränsningar. För energiintaget sätts även en övre begränsning, från början 120% av rekommenderat intag.

Dessa gränser och krav gör att utrymmet i det n-dimensionella rummet begränsas. Eftersom det finns både övre och undre begränsningar finns en risk att det inte finns några möjliga proportioner av receptet som uppfyller samtliga krav, det vill säga att det inte finns något acceptabelt utrymme i det n-dimensionella rummet. Om så är fallet höjs den övre begränsningen för energiintag med 10 procentenheter och mängden av ingredienserna med 50 procentenheter.

När ett utrymme med möjliga lösningar hittats kommer miljöaspekten in. Den punkt inom området som ger lägst klimatavtryck kommer att ses som den optimala punkten. Koordinaterna för denna punkt ger de nya mängderna av ingredienserna i receptet. För att förtydliga algoritmen beskrivs nedan ett antal av de ekvationer som används vid optimeringen med simplexmetoden.

Låt x_i representera antal gram av ingrediens i i det optimerade receptet, c_i klimatavtryck per gram av ingrediens i och n antal ingredienser i receptet. Målfunktionen blir då

$$min \quad c_1x_1 + c_2x_2 + ... + c_nx_n.$$

Låt ytterligare y_i vara antal gram av ingrediens i i originalreceptet, och p_{min} och p_{max} de procentsatser av respektive ingrediens som receptet ska innehålla som minst respektive mest, uttryckt i decimalform. Det ger följande bivillkor för mängden av respektive ingrediens:

$$\begin{aligned} p_{min}y_1 \leq x_1 \leq p_{max}y_1, \\ p_{min}y_2 \leq x_2 \leq p_{max}y_2, \\ \vdots \\ p_{min}y_n \leq x_n \leq p_{max}y_n. \end{aligned}$$

Genom att sedan låta f_i definieras som procentsatsen fett i ingrediens i, uttryckt i decimalform, och f_{need} behovet av fett för den aktuella användaren, fås även bivillkoret

$$f_1x_1 + f_2x_2 + ... + f_nx_n \geq f_{need}.$$

På samma sätt sätts bivillkor för energi, protein och kolhydrater.

4.4.3 Utvärdering av resultat

För att utvärdera resultaten av receptoptimeringen jämfördes ett antal recept från receptfavoriter.se [35] med deras optimerade motsvarighet. Optimeringen skedde med avseende på en standardanvändare med ett energibehov på 2 000 kcal. De värden som användes för jämförandet var klimatavtrycket från receptet per portion, samt värdet på L_2-normen (se avsnitt 2.3 Teoretiskt underlag och definitioner) där den optimala vektorn motsvarar ett recept som uppfyller 30% av användarens dagsbehov av fett, kolhydrater och protein.

Statistik hämtades både från en lista med godtyckliga recept och från en lista där recept som inte var lämpade för optimering hade filtrerats bort. Ett recept sågs inte som lämpat för optimering om det innehöll mindre än 50% av den mängd fett, kolhydrater eller protein som var rekommenderat för användaren i en måltid.

4.5 Menygenerering

Vid menygenereringen används recepten i receptdatabasen. Dessa recept är fixa, det vill säga inga proportioner i receptet kommer att ändras. Genereringen kan ske på tre sätt. I första fallet optimeras menyn med avseende på användarens näringsbehov, i andra fallet maximeras användandet av ingredienser som användaren har hemma och i tredje fallet minimeras klimatavtrycket från ingredienserna i menyn. Genereringen kan ge en meny med ett godtyckligt antal recept.

Processen börjar med en lista bestående av samtliga recept från receptdatabasen. De recept som innehåller ingredienser som användaren inte vill ha med, exempelvis på grund av allergi, plockas bort från listan. Därefter kan beräkningarna påbörjas. Vid genomgång av recept avläses hur många portioner respektive recept är beräknat för. Eftersom antal portioner varierar för olika recept skalas näringsvärdena från recepten om för en portion. Denna skalning görs för att kunna genomföra optimeringsberäkningarna för genereringen av menyn med avseende på användarens rekommenderade näringsintag. På samma sätt som vid receptoptimeringen antas en måltid utgöra 30% av det dagliga rekommenderade värdet. Näringsbehovet för hela menyn beräknas utifrån det värdet samt antalet recept i menyn.

4.5.1 Näringsoptimering

Parametern hälsa representeras här av rekommenderat intag av energi samt näringsämnen som fett, protein, kolhydrater, fibrer, ett antal vitaminer och ett antal mineraler. Värden för rekommenderade intag hämtas från användarprofilen. För att optimera det totala näringsintaget för ett antal recept, det vill säga en meny, kommer en beräkning av L_2-normen att användas på ett antal vektorer som representerar olika näringsämnen.

Utgångspunkten är en optimal vektor, $(1, 1, \ldots, 1)^T \in \mathbb{R}^n$, där n är antalet näringsämnen. Siffran 1 motsvarar 100% av rekommenderat intag av respektive näringsämne j,

$j = 1, \ldots, n$. Mängden av respektive näringsämne som menyn ger beräknas med hjälp av värden från livsmedelsdatabasen och receptdatabasen. Denna mängd divideras sedan med rekommenderat intag. Resultatet blir ett decimaltal som representerar den procentsats av rekommenderat intag som menyn ger. Beräkningen görs för respektive näringsämne vilket ger en ny vektor i $\mathbb{R}^n$. Målet är nu att minimera avståndet mellan ändpunkten på denna vektor och ändpunkten på den optimala vektorn. Här blir ekvationen för L_2-normen (se ekvation (2) i avsnitt 2.3) följande:

$$\min \sqrt{\sum_{j=1}^{n} (1 - p_j)^2}$$

där p_j motsvarar värdet på plats j i den uträknade vektorn för uppfyllt rekommenderat intag för näringsämne j. Om p_j överstiger 1, det vill säga 100%, sätts den ändå till 1. Anledningen är att rekommenderat intag för det aktuella näringsämnet då är uppnått. Även en övre begränsning för procentsatsen sätts för att undvika överdosering. Dessa värden hämtas från användarprofilen. Om något överdoseringsvärde överskrids sätts p_j till det faktiska värdet, och användaren meddelas om överdoseringen av näringsämnet.

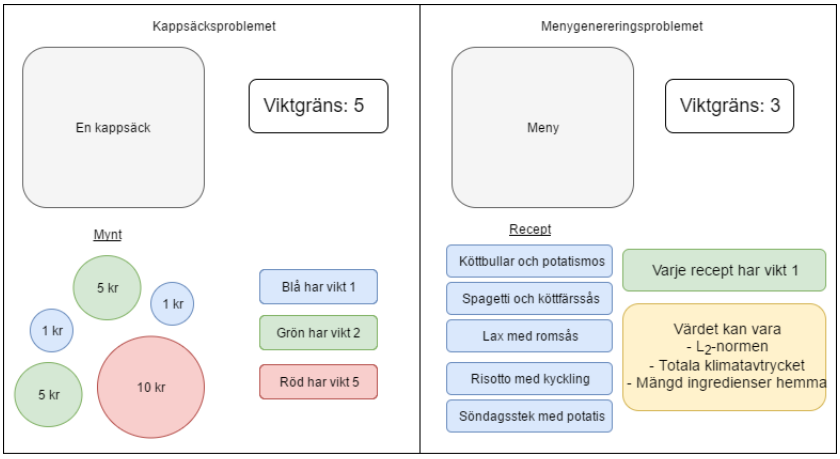
4.5.2 Ingrediensutnyttjande och klimatavtryck

När en meny ska genereras kan användaren ange en lista på vilka ingredienser hen har hemma. Användaren har sedan två alternativ att optimera efter. Det första är att räkna ut en meny som använder så mycket av användarens ingredienser som möjligt. Här subtraheras den mängd av ingredienser som används i menyn från användarens ingredienslista. Sedan jämförs resultatet för flera olika menyer och den meny som använder mest av ingredienserna som användaren har hemma returneras.

Det andra alternativet är att hitta den meny som ger lägst klimatavtryck. Här tas en inköpslista fram för varje meny. Den meny som ger en inköpslista med lägst klimatavtryck ses som den optimala. De ingredienser som användaren redan har hemma finns inte med på inköpslistan och anses alltså inte bidra till något klimatavtryck.

4.5.3 Algoritm grundad på dynamisk programmering

Till en början implementerades en algoritm som bygger på dynamisk programmering för att få ut olika kombinationer av recept utifrån en mängd recept. Menygenereringsproblemet identifierades under implementationsfasen som ett så kallat kappsäcksproblem [9]. Liknelsen med kappsäcksproblemet syns tydligt i figur 9 där vikten är antal recept och ett recept har vikten ett. Algoritmen itererar över en lista med recept från receptdatabasen genom rekursiva metodanrop där receptet som bearbetas skickas med, samt en lista med de sparade recepten till menyn, vilken är tom från början.



Figur 8: Ett bildexempel på ett kappsäcksproblem där målet är att maximera värdet av mynten i kappsäcken utan att överstiga viktgränsen.

Det finns två basfall. Det första basfallet är att listan med sparade recept innehåller det önskade antalet recept för menyn. Då jämförs listans värde med värdet för den för tillfället optimala menyn från föregående iteration. Ifall listan överträffar den för tillfället optimala menyn sparas istället den listan som optimal meny. Därefter returneras värdet för den nya optimala menyn. Andra basfallet är att recepten tar slut innan listan med sparade recept innehåller det önskade antalet recept för menyn. Då ignoreras den sparade listan och de rekursiva metodanropen avslutas. Ett sämre värde än det optimala returneras för att säkerställa att värdet ej blir optimalt.

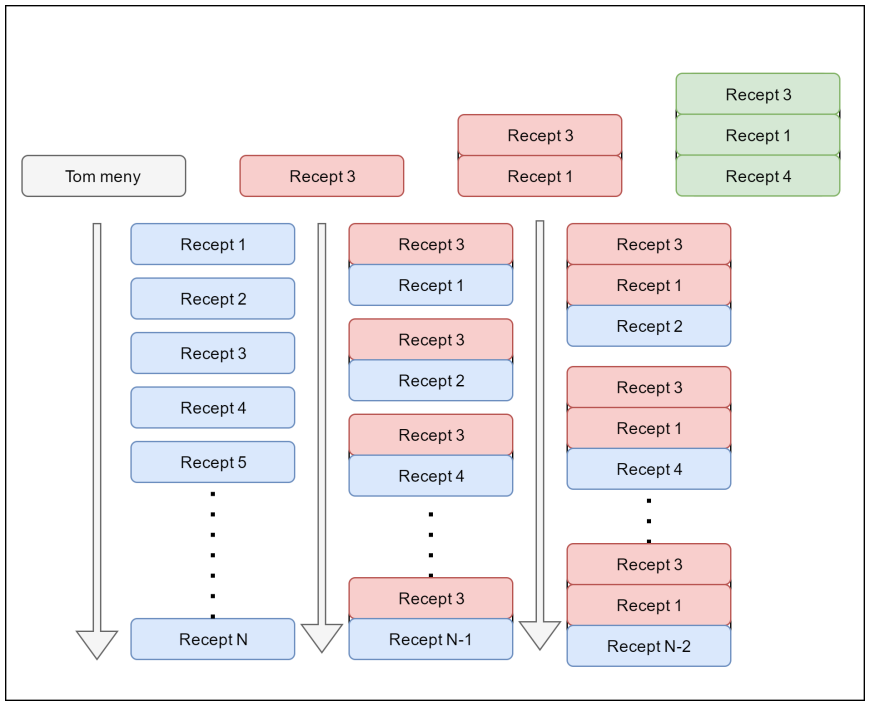
Om inget av basfallen uppfylls skapas två olika listor av den givna listan där den ena, kallad m_1, innehåller det nuvarande receptet, medan den andra, kallad m_2, ej innehåller receptet. Därefter görs två rekursionsanrop med varsin lista och med nästa indexnummer. Listan med lägst värde returneras enligt följande pseudokod:

```
menu(int n,array m_0) {
    if (m_0 har önskat antal recept)
        jämför m_0 med optimala menyn och spara den bästa
        returnera värdet av bästa menyn
    if (recepten tagit slut)
        returnera värdet av bästa menyn plus 1
    m_1 = m_0 + receptet med index n
    m_2 = m_0
    return min(menu(n-1, m_1), menu(n-1, m_2))
}
```

4.5.4 Girig algoritm

Mot slutet av projektet analyserades tidskomplexiteten av ovanstående algoritm och då upptäcktes att en misstolkning av dynamisk programmering hade gjorts. Tabelliseringen

och användningen av delresultat användes ej och därför hade algoritmen större exekveringstid än väntat. Därför började arbetet med att ta fram en snabbare algoritm, vilket resulterade i en så kallad girig algoritm. Även för denna algoritm är resultatet en meny med r antal recept. Den giriga algoritmen börjar med att analysera n recept för att hitta det bästa receptet r_1 som sedan läggs till i menyn *meny*. Receptet r_1 tas därefter bort från de övriga n recepten, vilket ger att det är *n-1* recept kvar som var och ett analyseras tillsammans med r_1. Resultatet blir ett par med r_1 och det recept som tillsammans med r_1 ger bäst värde. Sedan fortsätter ovanstående metod tills *meny* blir av storlek r. Kortaste avståndet till optimalt näringsintag (det vill säga lägst värde på L_2-normen), minst mängd rester eller lägst klimatavtryck avgör vilket som ger det bästa receptet, det bästa paret och så vidare. Algoritmen analyserar n recept första gången och $n-1$ recept den andra gången tills r recept finns i *meny*, vilket ger komplexiteten $O(n \cdot r)$. I figur 9 visualiseras ovanstående process.



Figur 9: En visualisering av hur den implementerade algoritmen fungerar, det vill säga hur flödet går för att hitta en meny bestående av tre recept.

4.5.5 Tidsanalys av algoritmerna

Två metoder användes för att observera programmets tidskomplexitet. Båda dessa använde sig av Javas tidsbibliotek. Först gjordes en grov estimering av hur många sekunder en uträkning tog. Vid analysen av 4.5.3 och 4.5.4 för olika antal önskade recept användes denna metod. Senare beräknades istället differensen i millisekunder mellan före och efter menygenerering. Denna metod automatiserades för ett intervall på 10 till 900 recept och

användes för att räkna ut skillnaden i observerad tidskomplexitet för olika antal recept i söklistan och olika antal önskade recept. Exempelvis vid 10 recept delades de 900 recepten in i 90 delar med 10 recept i varje och menygenereringen kördes 90 gånger. Den körning som tog längst tid noterades och fick stå för värstafallskomplexiteten. Denna metod användes tre gånger för ett, två respektive tio önskade recept i resultatet.

4.6 Användargränssnitt

För att redovisa resultaten och ta projektet ett steg närmare en applikation implementerades ett enklare användargränssnitt. Användargränssnittet är skrivet i HTML, CSS och JavaScript med hjälp av ramverket Vue.js för dynamik och strukur. Även Bootstrap användes i implementationen för att göra webbsidan responsiv och för att designen på gränssnittet ska vara enhetlig. Varje vy i gränssnittet består av en komponentfil innehållande design och funktioner för vyn. Navigeringen mellan de olika vyerna styrs med hjälp av en navigationsfil som de olika navigeringsknapparna anropar.

Kommunikationen mellan frontend och backend sker med hjälp av HTTP-anrop. Vid ett anrop använder sig backend av en routes-fil som distruberar vilken kontroller som ska anropas. Kontrollerna i sin tur kallar på de funktioner som ska exekveras. Det implementerades funktioner som returnerar data i form av JSON-objekt vilket underlättar hanteringen och visualiseringen av utskrifterna i användargränssnittet.

Funktioner och vyer som implementerades presenteras nedan.

- **Lägga till en användare.** Användaren fyller i all data som behövs för att systemet ska kunna genomföra receptoptimeringar och menygenereringar utifrån näringsrekommendationer, det vill säga information om ålder, kön, vikt och så vidare. Datan skickas till backend och sparas i användardatabasen som en användarprofil. Användarprofilen finns sedan tillgänglig som alternativ när recept optimeras eller menyer genereras.
- **Optimera recept.** Med tillagd användarprofil finns två huvudfunktioner tillgängliga i gränssnittet. En av funktionerna är att optimera valfritt recept från databasen. En användare anges och ett recept väljs från en scrollista. Ett HTTP anrop skickas till backend för att optimering ska ske. Gränssnittet presenterar sedan resultatet i ingredienser, mängd och annan intressant information så som kalorier, näringsfördelning och klimatavtryck för receptet.
- **Lägga till recept.** Det går även att lägga till ett recept som inte finns i databasen. Användaren anger då URL till valfritt recept från receptfavoriter.se [35]. Därefter kallar HTTP-anropen på parsningsfunktionen i backend och lägger sedan in receptet i receptdatabasen.
- **Generera meny.** Den andra funktionen är att användaren kan välja att generera en meny med valfritt antal recept och utifrån vald användarprofil. Ett anrop skickas till backend och genereringen sker med avseende på hälsa och utgår från de näringsrekommendationer användarprofilen har. Vid menygenerering finns funktionen att generera en tillhörande inköpslista för recepten i menyn. Inköpslistan presenteras i användargränssnittet med funktioner för att lägga till och ta bort livsmedel.

5 Resultat

I detta avsnitt presenteras resultat från projektets delar strängmatchning, receptoptimering, menygenerering och användargränssnitt. Resultat presenteras i form av exempel på hur olika receptsträngar har tolkats i parsningsprocessen, exempel på recept som optimerats och menyer som genererats. Också mer kvantitativa resultat som antalet rader som lyckats tolkas, kvantitativa mått på skillnader i näring och klimatavtryck i recept och uppmätta körningstider för menygenereringen. Kapitlet avslutas med bilder på användargränssnittet där motorns olika funktioner presenteras.

5.1 Strängmatchning mot livsmedelsdatabasen

Nedan presenteras resultat för tolkning och matchning av ingrediensträngar samt vad för data som går att få ut från till exempel en webbadress. Vid projektets slut byggde livsmedeldatabasen i grunden på Fineli varifrån 1 347 livsmedel och deras tillhörande näringsvärden kommer. Utöver dessa livsmedel kommer 56 från Livsmedelsverket och 22 från USDA.

5.1.1 Matchande av strängar

Nedan presenteras resultat för parsningsprocessen. Med rader menas antalet ingrediensrader som recepten består av. Ett recept har flera rader, en för varje ingrediens, 'en gul lök', 'en banan' och så vidare. Det är dessa rader som parsern hanterar en i taget.

Tabell 4: Resultat av receptparsning

Inlästa recept	**Antal lyckade recept**	**Antal lyckade rader**
1 268 recept	461 av 1 268	18 488 av 19 996

Av de rader som inte lyckades matcha på 1 268 recept beror samtliga av dessa på att livsmedlet inte finns i databasen. Det finns alltså inget livsmedel med en sökträng som matchar. Utöver det blir vissa matchade rader ändå inkorrekta. Eftersom manuell genomgång av nästan 20 000 rader inte är genomförbart inom en rimlig tidsram togs istället ett slumpmässigt stickprov på 300 ur de rader som lyckades matcha för att se huruvida de var helt korrekta.

Av 300 rader var 270 helt felfria, det vill säga korrekt vikt, antal och livsmedel var identifierat. Dessutom fanns all information i databasen för att kunna beräkna rätt näringsvärden. Av de 30 rader som inte var helt felfria innehöll en rad fel livsmedel, resten innehöll rätt livsmedel. Föregående nämnda rad var följande:

> '600 gram mört kött, helst filé, entrecote eller innerfile av nöt, gris eller lamm'

Fisken *Mört* finns i databasen och matchades istället.

Resterande 29 rader innehöll alltså rätt livsmedel men resulterade i andra komplikationer. Av dessa bestod åtta av rätt identifierade livsmedel samt mått och mängd, men i

databasen saknades information om hur enheten ska konverteras till vikt och därmed blir näringsberäkningarna för de strängarna felaktiga. Tre ingredienser identifierades med fel enhet ('stycken' istället för 'droppar'). 17 ingredienser identifierades korrekt men originalsträngen innehöll fler alternativa livsmedel som missades (1 krm vit- eller svartpeppar). En ingrediens hade fel antal: 4 bitar laxfilé på 140 gram styck blev 140 gram laxfilé.

5.1.2 Utdata

Det går att ge systemet en webbadress eller en textrad som ska tolkas. Svaret som ges är i stil med figuren nedan. Nedan syns att systemet bland annat identifierat ett livsmedel med id 121 från Fineli ur ett recept på carbonara med creme fraiche.


```
"title": "Carbonara med creme fraiche",
"portions": 4,
"energyKcalPerPortion": 1457,
"energyKcal": 5829,
"co2PerPortion": 1.395595,
"carbohydrates": 283,
"protein": 229,
"url": "https://receptfavoriter.se/recept/carbonara-med-creme-fraiche.html",
"ingredients": [
{
  "amount": {
    "quantity": 350.0,
    "unit": "GRAM"
  },
  "comment": null,
  "original": "350 gram spagetti",
  "kcal": 1155.2342256214147,
  "co2": 0.28,
  "food": {
    "id": 1142,
    "dataSource": "FINELI",
    "dataSourceId": 121,
    "name": "Spagetti",
    "group": "Spagetti",
    "tags": [

    ],
    "processing": "NO_TREATMENT",
    "category": "PASTA_AND_MACARONI"
  }
},
...
```

5.2 Receptoptimering

Klimatavtrycket per portion summerades för 200 recept från receptfavoriter.se [35] och även för deras optimerade motsvarighet. Resultatet blev att det totala klimatavtrycket minskade med 9,6 %, eller i genomsnitt 0,17 kg koldioxidekvivalenter per portion. Näringsvärdet enligt L_2-normen, vilket ska ligga så nära 0 som möjligt, gav i genomsnitt värdet 2,1 för originalrecepten och 1,4 för de optimerade recepten.

När endast de recept som från början uppfyllde 50 % av användarens behov av kolhydrater, proteiner och fett utvärderades minskade det totala klimatavtrycket med 50,0% eller i genomsnitt med 1,16 kg koldioxidekvivalenter per portion. Näringsvärdet enligt L_2-normen var för originalrecepten i genomsnitt 2,9 och för de optimerade recepten 0,6.

Nedan visas resultatet av optimeringen av ett recept på gravad laxpasta för standardanvändaren. Mängden lax, spagetti och senap ökas medan mängden av övriga ingredienser minskas.

Optimalt recept, Gravad laxpasta	
Gravad lax	*69,3 g*
Spagetti	*116,1 g*
Lätt crème fraiche	*48,6 g*
Senap	*12,1 g*
Honung	*2,7 g*
Citronsaft	*1,2 g*
Dill	*3,2 g*
Klimatavtryck:	*0,513 kg*
Energi:	*646 kcal*

Originalrecept, Gravad laxpasta	
Gravad lax	*50,0 g*
Spagetti	*75,0 g*
Lätt crème fraiche	*50,0 g*
Senap	*6,3 g*
Honung	*2,8 g*
Citronsaft	*1,3 g*
Dill	*3,3 g*
Klimatavtryck:	*0,422 kg*
Energi:	*463 kcal*

Samma optimering som ovan av ett recept på frasvåfflor ger nedanstående resultat. Observera att mängden vetemjöl ökar med drygt 200% medan mängden av resterande ingredienser minskas.

Optimalt recept, Frasvåfflor	
Vispgrädde	*69,3 g*
Vetemjöl	*181,9 g*
Vatten	*52,3 g*
Smör	*6,8 g*
klimatavtryck:	*0,441 kg*
Energi:	*966 kcal*

Originalrecept, Frasvåfflor	
Vispgrädde	*99,3 g*
Vetemjöl	*60,0 g*
Vatten	*75,0 g*
Smör	*9,8 g*
klimatavtryck:	*0,511 kg*
Energi:	*645 kcal*

Om ett recept på chili con korv optimeras för standardanvändaren fås följande resultat. Här har mängden socker ökats för att uppfylla kolhydratsbehovet.

Optimalt recept, Chili con korv	
Grillkorv (bratwurst)	*100,7 g*
Gul lök	*25,2 g*
Krossade tomater	*100,7 g*
Vita bönor i tomatsås	*220,5 g*
Socker	*40,3 g*
klimatavtryck:	*1,009 kg*
Energi:	*715 kcal*

Originalrecept, Chili con korv	
Grillkorv (bratwurst)	*100,0 g*
Gul lök	*25,0 g*
Krossade tomater	*100,0 g*
Vita bönor i tomatsås	*200,0 g*
Socker	*5,0 g*
klimatavtryck:	*0,898 kg*
Energi:	*447 kcal*

5.3 Menygenerering

Vid menygenerering skrivs vilken kombination av recept som är optimal för den givna användaren ut. Till att börja med skapas en lista av samtliga recept i receptdatabasen. I detta avsnitt är utgångspunkten en lista bestående av endast sex manuellt inskrivna recept. Att parsade recept inte används här beror på att fel livsmedel kan ha identifieras och därmed leda till felaktiga beräkningar. Genom att enbart använda manuellt inlagda recept säkerställs att de värden som hämtas till beräkningarna är korrekta. När listan är skapad genereras en meny med ett antal recept från listan. Användaren ställer in önskat antal recept i menyn, vilket i detta avsnitt är satt till tre.

5.3.1 Näringsoptimering

Nedan följer resultatet för standardanvändaren, med kaloribehovet 2 000 kcal. Den andra användaren är en person i projektgruppen. Det är en man på 30 år med längden 191 cm, vilket ger ett högre kaloribehov på ca 2 160 kcal. Samma optimeringsalgoritm som ovan ger för denna användare ett annat resultat.

Meny för standardanvändaren:	*Meny för andra användaren:*
Gravad laxpasta	*Gravad laxpasta*
Pokébowl med kyckling	*Hamburgare med pommes och coleslaw*
Chili con Carne	*Chili con Carne*

Näringsvärden i procent av användarens behov:

Energi	*100%*	*Energi*	*100%*
Kolhydrater	*81%*	*Kolhydrater*	*88%*
Protein	*100%*	*Protein*	*100%*
Fett	*100%*	*Fett*	*100%*
Fiber	*100%*	*Fiber*	*100%*
Vitamin A	*100%*	*Vitamin A*	*72%*
Vitamin B6	*100%*	*Vitamin B6*	*100%*
Vitamin B12	*100%*	*Vitamin B12*	*100%*
Vitamin C	*100%*	*Vitamin C*	*100%*

Vitamin D	*42%*	*Vitamin D*	*79%*
Vitamin E	*100%*	*Vitamin E*	*100%*
Tiamin	*100%*	*Tiamin*	*100%*
Riboflavin	*100%*	*Riboflavin*	*86%*
Niacin	*100%*	*Niacin*	*100%*
Folat	*100%*	*Folat*	*100%*
Kalcium	*28%*	*Kalcium*	*33%*
Fosfor	*100%*	*Fosfor*	*100%*
Kalium	*100%*	*Kalium*	*100%*
Magnesium	*100%*	*Magnesium*	*100%*
Järn	*100%*	*Järn*	*100%*
Jod	*43%*	*Jod*	*53%*
Selenium	*100%*	*Selenium*	*100%*
Klimatavtryck:	*0,393 kg*	*Klimatavtryck:*	*0,516 kg*

Här syns tydligt att resultatet för de två användarna skiljer sig åt. Första användarens meny saknar främst kalcium och vitamin D, medan andra användarens meny främst saknar vitamin A och jod.

5.3.2 Ingrediensutnyttjande och klimatavtryck

Nedan följer två menyer genererade från en lista av ingredienser som en potentiell användare kan ha hemma. Meny 1 är genererad i avsikt att ge ett så lågt klimatavtryck som möjligt och meny 2 för att använda så mycket som möjligt av ingredienserna som användaren har hemma.

Ingredienslista:

Röd paprika	*90,0 g*
Gul lök	*40,0 g*
Linser (röda, torkade)	*50,0 g*

Meny 1:

Gravad laxpasta
Frasvåfflor
Linsgryta

Mat som inte användes:

Röd paprika	*52,5 g*
Gul lök	*15,0 g*
Linser (röda, torkade)	*5,0 g*

Klimatavtryck: 1,362 kg

Meny 2:

Pokébowl med kyckling
Chili con Carne
Linsgryta

Mat som inte användes:

Röd paprika	*2,5 g*
Linser (röda, torkade)	*5,0 g*

Klimatavtryck: 6,639 kg

5.3.3 Algoritmanalys

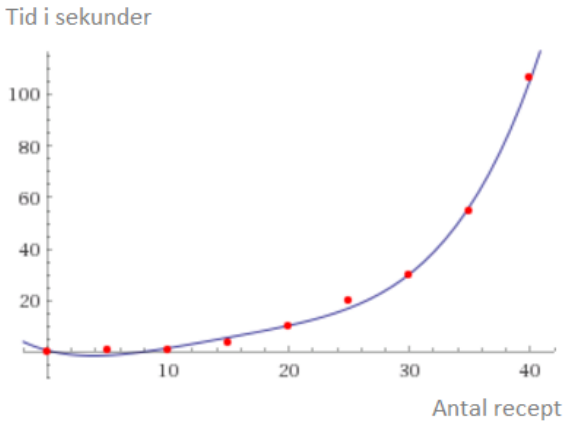
Tillvägagångssättet för följande algoritmanalys finns förklarat i avsnitt 4.5.5. Utifrån denna metod kan följande värden observeras.

Dynamiska programmeringsalgoritmen (Metod 4.5.3):
Tiden för menygenerering med mellan 5 och 40 recept i söklistan och tre recept som resultat mättes med hjälp av Javas tidsbibliotek. Följande värden observerades:

Tabell 6: Tid i sekunder för generering av meny med tre recept

Totalt antal recept	**Tid [sekunder]**
5	1
10	1
15	4
20	10
25	20
30	30
35	55
40	106

Inmatning av datapunkterna samt applicering av en så kallad quartic fit genom WolframAlpha [36] ger följande graf:



Figur 10: Ekvation uppskattad genom least squares fitting: $0.000169231x^4 - 0.00959907x^3 + 0.208683x^2 - 1.20594x + 0.780886$.

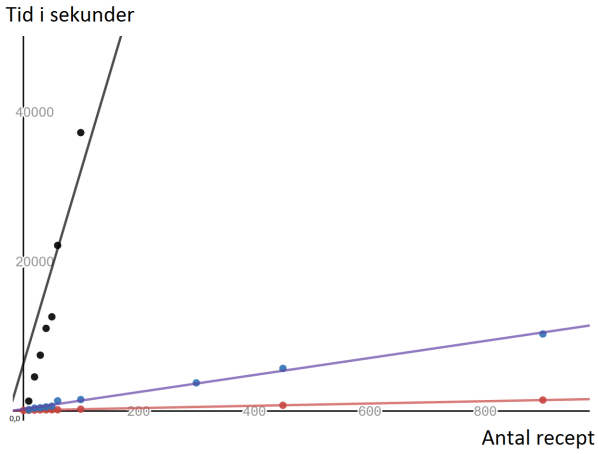
Giriga algoritmen (Metod 4.5.4):
Resultaten för den andra algoritmen har två synvinklar. Den ena är vad som händer om totala antalet recept i söklistan ökas eller minskas, medan andra är vad som händer när antalet önskade recept i den resulterade menyn ökas eller minskas.

Tider för menygenereringen med olika antal önskade recept i menyn samt olika antal recept i söklistan finns representerade i tabellen nedan.

Tabell 7: Tidtabell för menygenerering, där r är antalet önskade recept i menyn. ms står för millisekunder och s för sekunder.

Totalt antal recept	**r=1**	**r=2**	**r=10**
10	66 ms	162 ms	1,2 s
20	87 ms	353 ms	4,5 s
30	118 ms	406 ms	7,4 s
40	121 ms	519 ms	11,0 s
50	135 ms	620 ms	12,6 s
60	133 ms	1329 ms	22,1 s
100	219 ms	1504 ms	37,2 s
300	960 ms	3,7 s	100,4 s
450	732 ms	5,7 s	153,6 s
900	1,4 s	10,2 s	218,9 s

Tiden för menyalgoritmen från 10 till 900 recept i söklistan och de tre kolumnerna ovan ger följande graf:



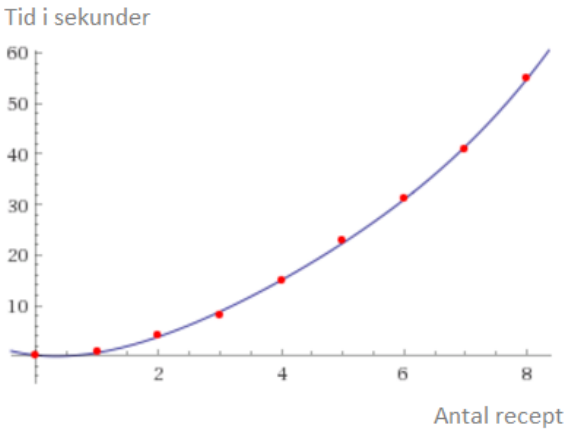
Figur 11: Ovanstående graf avspeglar tabell 7. Den röda linjen representerar resultatet för ett recept, den blåa för två recept och den svarta för 10 recept.

Tiden för menygenereringen med 450 respektive 996 recept i söklistan och med 1-8 recept i menyn uppmättes med hjälp av Javas tidsbibliotek. Följande medelvärden observerades:

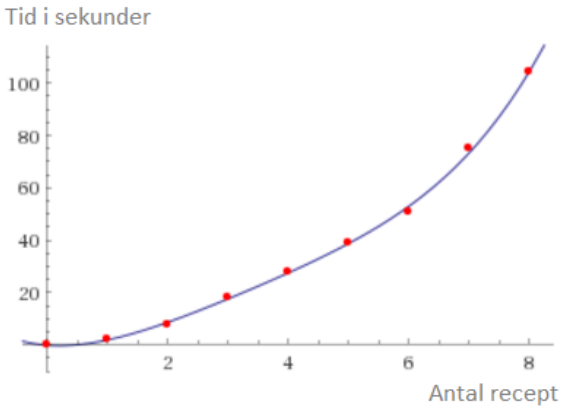
Tabell 8: Tiden i sekunder för uträkning med den giriga algoritmen för totalt 450 respektive 996 recept

Antal önskade recept	**Tid för 450 recept**	**Tid för 996 recept**
1	1 s	2 s
2	4 s	8 s
3	8 s	18 s
4	15 s	28 s
5	23 s	39 s
6	31 s	51 s
7	41 s	75 s
8	55 s	104 s

Inmatning av datapunkterna för 450 respektive 996 recept samt applicering av en så kallad quartic fit genom WolframAlpha [36] ger följande figurer:



Figur 12: Graf för 450 recept från WolframAlpha, med den approximerade ekvationen $0.0603147x^4 - 0.79921x^3 + 4.37869x^2 - 1.69807x - 0.047397$.

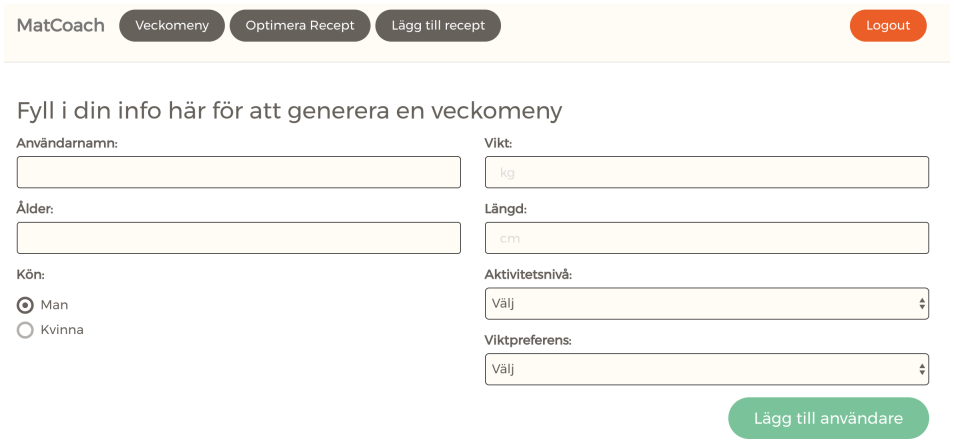


Figur 13: Graf för 996 recept från WolframAlpha, med den approximerade ekvationen $0.0151515x^4 - 0.239057x^3 + 1.95202x^2 - 1.24327x + 0.161616$.

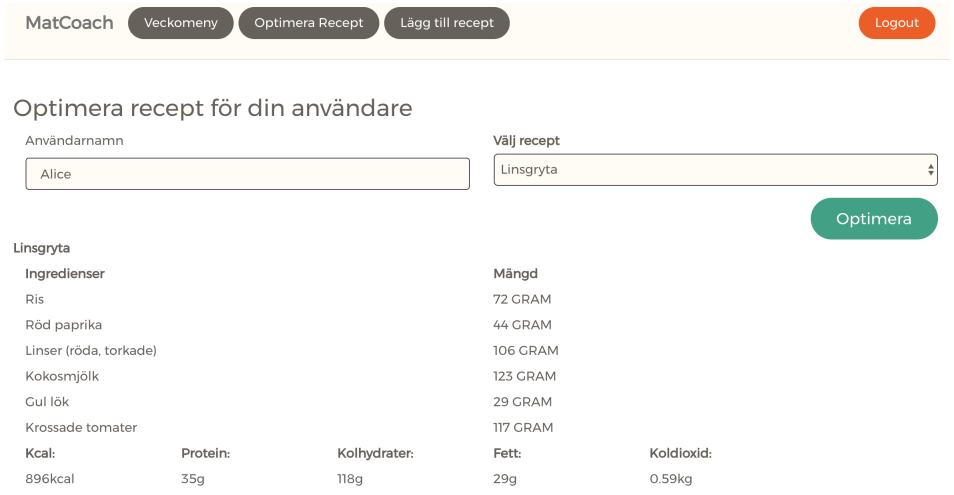
5.4 Användargränssnitt

Användargränssittet resulterade i följande vyer

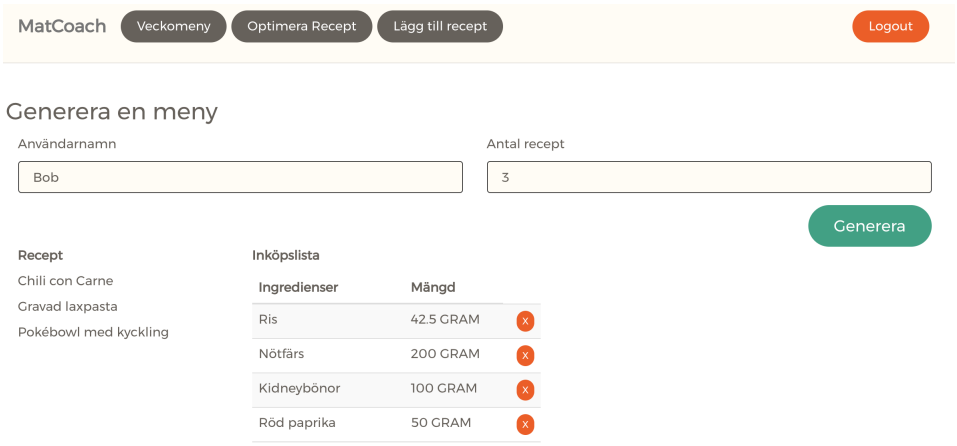
- **Välkomstsida.** När en användare besöker sidan möts denne av en välkomstsida som kort förklarar vad applikationen har för funktioner och syfte.
- **Lägg till användare.** I denna vy kan en användare göra en användarprofil genom att fylla i användarnamn, vikt, längd, kön, aktivitetsnivå och om användaren vill bibehålla, minska eller öka i vikt. Se figur 14.
- **Optimera recept.** Användaren anger en användarprofil och recept genom en scrollista. Gränssnittet presenterar ingredienser, mängd, näringsfördelning, kalorier och klimatavtryck för receptet. Se figur 15.
- **Lägg till recept.** Användaren anger en URL i skrivfältet och trycker på 'Lägg till', en bekräftelse på att receptet lyckades parsas visualiseras.
- **Generera meny.** Användaren anger användarprofil samt antal recept. Gränssnittet presenterar menyn med tillhörande inköpslista. Se figur 16.



Figur 14: 'Lägg till användare'-vy



Figur 15: Receptoptimeringsvy



Figur 16: Menygenereringsvy

6 Diskussion

Nedan utvärderas och diskuteras resultaten av de olika delarna av projektet. Tillvägagångssätten för strängmatchning, receptoptimering och menygenerering med tillhörande tidskomplexitet motiveras. Sist i kapitlet föreslås möjliga förbättringar av motorn.

6.1 Strängmatchning mot livsmedelsdatabasen

Uppgiftens svårighet låg i att textsträngar skrivna av människor inte alltid skrivs på enkel form. Alla strängar med stilen '1 msk olivolja' gick som förväntat smärtfritt tidigt i processen medan strängar som '2 nypor svart- eller vitpeppar' inte kunde hanteras av den enkla modell som var tilltänkt från början. Arbetet på en mer sofistikerad process gav upphov till att vissa saker gick bra medan andra slutade fungera. Det var helt enkelt svårt att matcha alla fall. I förlängningen handlar det om att få en dator att analysera en isolerad del av det svenska språket och få den att fatta beslut grundat på denna analys.

Dessutom behövde livsmedelsdatabasen hela tiden utökas med mer data för att förbättra tolkningsprocessen. Saker som densitet och vikt för enstaka livsmedel behövde läggas till för många livsmedel för att näringsberäkningen skulle fungera så bra som möjligt. När det var gjort upptäcktes att det även kunde behövas värden för vad ett paket väger, vad en klyfta väger och så vidare.

Maskininlärning var något som övervägdes som tillvägagångssätt för parsningsprocessen. I början gjordes tester med färdiga bibliotek för att se vilka resultat det kunde ge. Det visade sig att det krävdes väldigt mycket tid för att märka inlärningsdata. Projektgruppen lyckades inte hitta träningsdata på svenska för ingredienser. Även när träningsdata på engelska översattes grovt med hjälp av översättningsverktyg hade algoritmen ofta svårt att identifiera enkla saker som antalet eller enheten i en sträng. Även om ett maskininlärningssystem är mer sofistikerat än det slutligen implementerade tillvägagångssättet bedömdes problemdomänen vara så pass simpel att det skulle innebära mer problem än hjälp att experimentera med maskininlärning. Valet föll därför på simplare strängmanipuleringsmetoder.

Resultatet var väntat, alla ingredienser som hade en referens i livsmedelsdatabasen och dess strängar gick att parsa. Stickprovet visar att endast en sträng av 300 innehöll ett feltolkat livsmedel, vilket är ett gott resultat. Det fanns dock andra mindre problem. När en ovanlig enhet som 'droppar' inte finns bland listan på enheter hittas den inte vilket gör att strängen får 'styck' som enhet, som beskrivet under metod. Detta gör i sin tur att parsern frågar databasen efter hur mycket 'en Tabasco' väger, vilket det saknas värde på. Problemet går att lösa genom att lägga till fler enheter i systemet.

En annat problem var avsaknaden av alternativa livsmedel i 17 strängar. Dessa innehöll ett korrekt identifierat livsmedel men innehöll också till exempel ordet 'eller' och sedan ett antal alternativa livsmedel, vilka inte var identifierade. Mot slutet av projektet experimenterades med att låta parsern hitta alternativ i en sträng, men eftersom det endast fanns stöd för identifiering av ett extra livsmedel och inte ett godtyckligt antal

bedömdes dessa strängar som ej korrekt identifierade under stickprovet. Det finns även andra undantagsfall som 'vit- eller svartpeppar' som inte heller lyckades.

Ytterligare något som är värt att uppmärksamma är hur den mänskliga faktorn kan påverka resultatet. Tillägg av sökströngar görs manuellt och ger därför utrymme för misstag samt mänskliga värderingar. När standardlivsmedel bestäms är det oundvikligt att en människa tar ett beslut grundat på sina egna värderingar. Likaså när sökströngar läggs till. Ska den unika sökströngen 'köttfärs' hänvisa till nötfärs, fläskfärs eller blandfärs? Det är förvisso ett senare problem vars konsekvenser kommer visa sig när den potentiella applikationen faktiskt används, men att vara medveten om det när strukturen byggs är väsentligt.

Under testerna av antalet lyckade rader observeras att andelen lyckade recept får ses som låg medan andelen lyckade rader ser betydligt bättre ut. Det betyder att en majoritet av de recept som inte lyckas hitta ett livsmedel för alla rader i sin ingredienslista endast misslyckas med en eller möjligen två rader, resten går bra. Även om över 90% av ingredienserna i ett recept lyckas matchas så betraktas receptet som misslyckat ifall sista raden inte lyckas. Det beror som tidigare nämnts på att livsmedelsdatabasen inte är komplett med alla livsmedel som kan tänkas befinna sig i ett recept. Att fylla och formatera data till databasen är ett tidsödande arbete och inget som lagts stor vikt vid under projektet. Snarare har tid lagts på att bygga vägar för inmatning av data, vilket har gett ett gott resultat eftersom tre stora databaser stöds.

För att sammanfatta så går det med säkerhet att påstå att ingrediensströngar som har ett motsvarande livsmedel i livsmedelsdatabasen kommer att parsas, men att det inte alltid kommer vara helt korrekt på grund av ohanterade undantagsfall. Detta baseras på det slumpmässigt utvalda stickprovet på 300 rader. Slutligen kan nämnas att så gott som alla av de parsade strängarna innehåller ett korrekt identifierat livsmedel (299 av 300) samt att en absolut majoritet dessutom översatt all information i strängen till fullt användbar näringsinformation (270 av 300). Systemet är robust och klarar av att hitta ingrediens, mängdenhet och kvantitet samt eventuella textkommentarer oavsett i vilken ordning det är skrivet i de allra flesta fall.

6.2 Receptoptimering

Som nämnts i avsnitt 4.4.2 skalas recepten om efter användarens kaloribehov. Skalningen görs för att få personligt anpassade recept med lagom stora portioner. Bivillkoren som beskrivs i avsnitt 4.4.2 är valda för att ge så bra anpassade recept som möjligt. Anledningen till att recepten ska innehålla 75-150% av varje ingrediens är att de inte ska komma för långt från originalreceptet och för att minska risken för konstiga smakkombinationer. Värdena på gränserna är uppskattade efter vad gruppen ansåg rimligt. De nedre gränserna på fett, kolhydrater och protein är satta för att användaren ska få balanserade recept och lagom stora portioner. Utan dessa skulle algoritmen ge för små portioner eftersom det ger ett lägre klimatavtryck. Den övre gränsen på kaloriinnehåll finns för att algoritmen inte ska välja för mycket av en ingrediens med lågt klimatavtryck men högt kaloriinnehåll.

Om bivillkoren resulterar i att det inte finns någon lösning ändras de. Det kan visserligen göra att receptet får för högt kaloriinnehåll eller att proportionerna av ingredienserna är mycket olika de i originalreceptet men det gör också att algoritmen hittar ett recept som är så optimalt som möjligt även om det ursprungliga receptet är näringsmässigt obalanserat. Ett exempel på det är recepten på frasvåfflor och chili con korv som optimerats i avsnitt 5.2.

Algoritmen tar inte hänsyn till användarens behov av vitaminer och mineraler. Anledningen till det är att om de sätts till bivillkor på samma sätt som fett, kolhydrater och protein går det oftast inte att hitta en lösning. Det prioriterades därför inte att låta algoritmen ta hänsyn till vitaminer och mineraler. Eftersom näringsintaget under en dag kommer från en kombination av olika maträtter, och inte bara från en måltid, antas vitaminer och mineraler kombineras någorlunda ändå. Ett möjligt sätt att ta hänsyn till fler näringsämnen i algoritmen är att lägga till ett viktat värde av näringsintaget uträknat med L_2-normen på samma sätt som i målfunktionen i 4.5.1. En annan brist är att algoritmen inte tar hänsyn till vilken typ av kolhydrater och fett ingrediensen innehåller. Det gör att till exempel andelen socker i ett recept kan ökas för att uppfylla användarens behov av kolhydrater, se recept chili con korv i avsnitt 5.2.

I de flesta fall verkar optimeringen ge nya mängder av samtliga ingredienser, men med inte allt för stor skillnad från originalreceptet (se exempelvis optimeringen av receptet på gravad laxpasta), vilket var tanken med funktionen. En observation är att samtliga optimerade recept i avsnitt 5.2 har ett högre klimatavtryck än originalreceptet. Det beror på att portionsstorleken har ökats; i receptet på gravad laxpasta för att ursprungsportionerna var för små och i de andra två recepten för att de från början var obalanserade. Ett annat problem med de optimerade recepten är att förhållandet mellan vätska och fasta ingredienser kan ändras mycket. Vid optimeringen av receptet på frasvåfflor tredubblas mängden vetemjöl, medan mängden av både vispgrädde och vatten minskar. Det skulle resultera i en väldigt fast och svåranvänd våffelsmet.

De kvantitativa testerna, vars resultat presenteras i avsnitt 5.2, visar att de optimerade recepten i genomsnitt ger både bättre näringsinnehåll och lägre klimatavtryck. Särskilt bra fungerade optimeringen för de recept som från början var någorlunda balanserade näringsmässigt. Värt att tänkas på är att testerna inte tar hänsyn till vilken typ av kolhydrater och fett som används, inte heller innehållet av vitaminer och mineraler testades. Slutsatsen kan därför inte dras att recepten blir mer hälsosamma när de optimeras, även om det finns resultat som tyder på det.

6.3 Meny med näringsoptimering

Hur mycket av användarens behov av olika näringsämnen som uppfylls för en meny skrivs ut enligt tidigare presenterade resultat, se avsnitt 5.3.1. Målet var att så många som möjligt av dessa behov skulle uppfyllas. Det gjordes genom att minimera L_2-normen. Anledningen till valet av L_2-normen, och inte exempelvis L_1-normen, är att värdena för näringsämnena generellt hamnar närmare det optimala med L_2-normen. Med L_1-normen

hade risken varit större att värdena för vissa näringsämnen hamnat långt ifrån det optimala om flera andra näringsämnen hamnar väldigt nära. Risken för det är alltså lägre med L_2-normen, och ger istället jämnare värden.

Kom ihåg målfunktionen med L_2-normen från avsnitt 4.5.1:

$$\min \sqrt{\sum_{j=1}^{n} (1 - p_j)^2}.$$

Om behovet av ett näringsämne är uppfyllt, och samtidigt inte överskrider överdoseringsvärdet, sätts värdet till 100%, det vill säga $p_j = 1$ för näringsämnet. Denna term bidrar alltså inte till summan i L_2-normen. Vid optimeringen kommer därför mängden av de näringsämnen vars behov inte är uppfyllt försöka höjas, snarare än att mängden av näringsämnet vars behov redan är uppfyllt ökas, eftersom termen för ett uppfyllt näringsbehov redan är minimal, det vill säga 0. Bakomliggande argument till denna metod är att så många näringsvärden som möjligt ska uppnå rekommenderat intag, men samtidigt inte överdoseras.

Flertalet av näringsbehoven för de två genererade menyerna under Resultat (avsnitt 5.3.1) ligger på 100%. I detta fall verkar alltså algoritmen med L_2-normen ge ett bra resultat som uppfyller den funktion som eftersträvades. Slutsatsen om ett gott resultat kan dock inte dras generellt eftersom algoritmen endast är testad för ett fåtal olika fall, det vill säga med avseende på olika användare, antal recept i menyn och en viss receptdatabas. Vår databas bestod här enbart av sex recept. Med fler recept i databasen bör ännu fler näringsbehov uppfyllas eftersom fler recept ger fler möjliga kombinationer av näringsämnen för en meny, vilket ger större möjlighet att uppfylla näringsbehoven.

De genererade menyerna i avsnitt 5.3.1 visar också att menyn är anpassad efter användaren och hens behov. Ett recept skiljer sig åt mellan menyerna för standardanvändaren och den 30-åriga mannen, trots endast sex recept att välja på. Menyerna saknar en viss mängd av vissa näringsämnen, dock med viss variation för de olika menyerna. Denna variation är logisk eftersom menyerna inte innehåller exakt samma recept. I detta fall är alltså menyn personligt anpassad, vilket var ett av delmålen med denna menygenerering. Dock kan inte slutsatsen om personligt anpassade menyer dras eftersom algoritmen, återigen, endast är testad för ett begränsat antal fall.

6.4 Meny med ingrediensutnyttjande och klimatavtryck

De två funktionerna som genererar menyer med maximalt användande av en ingredienslista eller minimalt klimatavtryck ger bra resultat i det område de är ämnade för. Däremot samverkar funktionerna inte på något sätt. Därför kan en meny som använder mycket av ingredienserna som användaren har hemma ge ett högt klimatavtryck. Det syns tydligt i resultaten i stycke 5.3.2 och 5.3.2 där samma indata resulterar i en meny med 1,362 kg koldioxidekvivalenter och en med 6,639 kg. I det fallet kan det vara önskvärt att istället för att använda så mycket som möjligt av ingredienserna som användaren har hemma, bara använda en del av ingredienserna och istället ge en meny med lägre

klimatavtryck.

Eftersom algoritmen som minimerar klimatavtryck också tar hänsyn till vad användaren har hemma kan den ge bra resultat i båda avseendena. Det är dock inte säkert att menyn innehåller någon av ingredienserna som användaren har hemma eftersom den eventuellt kan hitta recept med lägre klimatavtryck. Det kommer att hända om det finns några recept i databasen med mycket lägre klimatavtryck än andra. En annan följd av det är att algoritmen ofta ger ut samma meny. En möjlig lösning skulle vara att lägga ihop värdena som används i de två optimeringarna och vikta dem mot varandra.

6.5 Algoritmanalys

Projektets motor har en linjär tidskomplexitet vid menygenerering med avseende på mängden recept i söklistan, men en icke-linjär komplexitet när det gäller antal önskade recept i menyn. Tidskomplexiteten för den giriga menyalgoritmen beräknades i metod 4.5.4 till $O(n \cdot r)$, där n är totala antalet recept i söklistan och r är antalet önskade recept i menyn. Resultatet i avsnitt 5.3.3 visar att för antal recept i den resulterande menyn är tidsökningen polynomiell av fjärde graden med avseende på antal önskade recept (se figur 12 och 13), men för antal recept i söklistan är tidsökningen linjär (se figur 11). Tidskomplexiteten blir i så fall $O(n \cdot r^4)$. Dock syns i resultatet i figur 11 att värdena är ganska ojämna, vilket beror på att alla recept är olika stora och därför kräver olika lång tid för att jämföras. När komplexitetsanalysen genomfördes var det stor variation mellan olika recept och därför blev komplexiteten olika för olika recept, vilket visas i figur 11 i resultat 5.3.3. För att säkerställa hypotesen att receptens storlek avgör komplexiteten skulle ytterligare tester på recept med lika stor ingredienslista kunna matas in och testas. Tyvärr visar figuren också att komplexiteten växer sig allt för stor redan vid 10 recept i menyn, vilket även visas i tabell 7. Då är den enda lösningen att satsa på att få recept med bra värden och endast då kan menygenereringen fungera i realtid även för stora menyer.

Dynamisk programmering fungerar inte för näringsberäkning på det sätt som först var tänkt. Menygenereringsproblemet tolkades först att vara av samma karaktär som kappsäcksproblemet, som nämnts i metod 4.5.3. Resultatet visar att tidskomplexiten för försöket med dynamisk programmering är allt annat än optimal. Under teori 2.2 nämns att en dynamisk programmering behöver spara värden som sedan används av kommande dellösningar. Det är inte möjligt med L_2-normen eftersom den inte följer den distributiva lagen. Alla kombinationer av recept i både antal och storlek ger olika värden. I kappsäcksproblemet är alla värden distributiva. Därför utvecklades istället ovan nämnda giriga algoritm för att kunna lösa problemet. Däremot skulle menygenerering med avseende på klimatavtryck kunna fungera eftersom värdet följer distributiva lagen och kan därmed spara och använda dellösningar som dynamisk programmering kräver.

6.6 Användargränssnitt

Ett välutvecklat användargränssnitt var aldrig syftet eller något som prioriterades i projektet. Det lades därför inget fokus på design och användarvänlighet. Målet var att skapa

en frontend som kunde presentera grundfunktionerna i systemet ifall tiden räckte till. Annars skulle resultatet presenteras via terminalen. Därför var många resultatutskrifter från funktionerna i backend i textformat och korrigeringar var i behov. Det saknades också vissa funktioner på backend för att få ett funktionellt gränssnitt, vilka också implementerades. Trots tidskrävande korrigeringar resulterade användargränssnittet över projektets mål.

6.7 Utvecklingspotential

Systemet är inte helt färdigt och har ett flertal områden med utvecklingpotential. Nedan listas några förslag på möjliga utvecklingar av systemet.

6.7.1 Strängmatchning mot livsmedelsdatabasen

Som nämndes i avsnitt 6.1 är det brist på livsmedel i livsmedelsdatabasen. Ohanterade undantagsfall gör att ingrediensssträngar kan bli parsade på ett inkorrekt sätt. I ett potentiellt fortsatt arbete kan livsmedel eventuellt hämtas från andra länders motsvarande livsmedelsdatabaser, eller läggas till manuellt. Strukturen finns och är möjlig att bygga på.

När det gäller ohanterade undantagsfall bör strängar som innehåller ordet 'eller' hanteras på ett flexibelt och stukturerat sätt. Den aktuella implementeringen hanterar ej sådana strängar på ett sådant sätt, vilket är problematiskt eftersom ordet förekommer relativt ofta i ingrediensssträngar. Dessa strängar står i dagsläget för en stor del av strängarna som parsas på ett felaktigt sätt. De är komplexa för att ordet 'eller' används på många olika sätt i olika strängar och kan därför inte hanteras på samma sätt. Sättet motorn i dagsläget löser 'eller'-problematiken på är att den letar efter ett livsmedel och tar det första som hittas i ingrediensssträngen och lägger till 'eller' och den resterande strängen i kommentaren. Denna lösning gör dock att information förloras och indirekt att ett ogenomtänkt beslut tas om vilket av livsmedlen som ska användas, därför är inte lösningen tillräckligt stabil.

Ett förslag på hur 'eller'-problematiken skulle kunna lösas är att först undersöka om ingrediensssträngen innehåller 'eller'. Om den gör det görs ett antagande att strängen innehåller två eller fler ingredienser som det är möjligt att välja mellan när receptet ska tillagas. Första delen av strängen innan ordet 'eller' skickas in i parsningsprocessen och hanteras som en egen ingrediens. Efter det skickas delen av strängen efter 'eller' in i parsningsprocessen. Om den delen av strängen inte har någon mängd görs ett antagande att den har samma mängd som strängen innan 'eller' hade. Strängar som innehåller 'eller' blir alltså två matobjekt som sparas. Dessa matobjekt sparas för att användaren i framtiden ska kunna göra valet mellan de två olika ingredienserna själv, och då ska information om båda dessa finnas. Detta tillvägagångssätt experimenterades med delvis under projektets slut och är nämnt tidigare under 6.1. Förslaget är dock som sagt inte komplett. Till exempel kommer strängar som '1 msk svart -eller vitpeppar' inte gå att parsa med en sådan implementering eftersom att 'svart' inte är något livsmedel. Hanteringen av det här undantagsfallet är något som bör prioriteras i fortsättningen, för att rätt livsmedel ska visas utan att information förloras.

Autocorrect bör ses över och eventuellt tas bort. Här kan det vara lämpligt att ta fram och titta på statistik över hur ofta funktionen faktiskt gör så att ett felstavat livsmedel identifieras samt hur ofta ett livsmedel blir felaktigt identifierat på grund av funktionen. Dessa observationer får sedan ställas mot varandra för att ett motiverat beslut ska kunna tas.

API:t JSON tagger som används för att identifiera ordklasser samt nummer i parsningsprocessen var från början tänkt att ta större plats än vad det gör i nuläget. Det API:t används till i dagsläget är att identifiera nummer i söksträngen. Det finns enklare, mindre tidskrävande och bättre sätt att göra det på direkt i javakoden, och därför kan det vara bättre för processen om API:t inte kallas på överhuvudtaget. Det är påfrestande att kalla på ett externt API för varje ingrediens i varje recept om många recept ska parsas. Det är dessutom alltid säkrare att vara oberoende av källor som är utom ens kontroll. Däremot kan det finnas fördelar med att behålla API-anropen för att kunna identifiera och byta ut ord i framtiden.

Ytterligare något som kan och bör förbättras är kommentarstrukturen. I nuläget läggs allt som inte är livsmedel, mått eller kvantitet i en gemensam kommentarsträng. Metoden fungerar men kommentaren blir inte en snygg och sammanhängande mening, vilket är något som kan ses över för att användaren ska få en så bra upplevelse som möjligt.

Densitet för livsmedel som finns för att kunna omvandla enheter ska ligga i livsmedelsdatabasen, men i dagsläget saknas densiteten på många av livsmedlen. Bristen på densitetvärden är ett problem eftersom en korrekt omvandling är viktig för att näringsberäkningarna ska stämma. Problem måste ses över för att motorn ska kunna vara användbar och ge korrekta beräkningar.

6.7.2 Användarprofil och användargränssnitt

Potentiella förbättringar för användarprofilen är många. Det naturliga nästa steget går hand i hand med frontend-delen i form av att skapa ett login-system där det krävs lösenord för att få del av en användares information. Varje unik användare skulle då sparas i en databas med all sin personliga data. Denna utveckling kräver också att datasäkerheten ökas med krypterade lösenord och standardlösningar mot vanliga attacker som kan läcka känslig information.

Ytterligare en möjlighet är att implementera en skala på vilka prioriteringar användaren vill ha på optimeringarna. Istället för att algoritmen bara optimerar utifrån näringsintag, klimatavtryck eller ingrediensutnyttjande, kan användaren ställa in en optimering som tar alla tre parametrar i synpunkt, och själv bestämma genom en skala vilka parametrar som är viktigast för hen. Det förutsätter att förbättringsförslaget i avsnitt 6.7.5 (Menygenerering för fler parametrar) genomförs.

Att spara användarens geografiska plats skulle kunna bidra till intressanta extrafunktioner för optimering med avseende på ekonomi och möjligen miljö. Applikationen skulle

då kunna rekommendera varor på extrapris i närliggande butiker. Priset skulle kunna minimeras och klimatavtrycket skulle kunna minskas genom att varor som produceras nära användarens geografiska plats rekommenderas.

En annan förbättring skulle kunna vara att implementera funktionen att användaren ska kunna lägga in egna recept, kunna bli vän med andra användare i applikationen, chatta med dem, byta recept och samla poäng beroende på exempelvis hur miljövänligt användaren äter.

6.7.3 Hänsyn till smak vid utbyte av ingredienser

Tanken från början var att motorn skulle kunna byta ut ingredienser i ett recept optimerat på näring och klimatavtryck, med hänsyn till användarens preferenser. Det är en möjlig förbättring att implementera. Något som blir problematiskt är att ingen hänsyn tas till människans smaksinne. Det kan resultera i att ett ingrediensbyte som optimerar på näringsvärden ger en ingrediens som enligt människans smaksinne inte passar ihop med resten av receptets ingredienser. För att lösa det borde människans smaksinne analyseras och sedan bör metoder skapas med resultatet av analysen i åtanke. Att problemet löses är väsentligt för att utbytet av ingredienser ska vara användbart.

Ett förslag på en åtgärd är att skapa en typ av kategoriseringsmodell för recepten i receptdatabasen, för att sedan undersöka frekvensen av när en ingrediens förekommer tillsammans med andra specifika ingredienser inom en viss kategori. Görs denna analys på en stor mängd data finns det sedan data på vilka ingredienser som förekommer ofta tillsammans. Ingredienser som ofta förekommer tillsammans kan antas passa bra ihop och därför skulle datan kunna användas till att byta ut en ingrediens. Processen skulle förslagsvis kunna gå till så att programmet får en önskan om att en viss ingrediens ska bytas ut. Då undersöks vilka ingredienser denna ingrediens ofta förekommer tillsammans med. I nästa steg undersöks de ingredienserna och vilka ingredienser som ofta förekommer tillsammans med alla ingredienser i receptet, förutom just den ingrediensen som ska bytas ut. De ingredienser som bäst uppfyller dessa kriterier skulle då bli kandidater till ingredienser som kan bytas in i receptet. Förhoppningsvis skulle exempelvis fläsk kunna bytas ut till bacon eller skinka i pasta carbonara, eftersom bacon, fläsk och skinka används på samma sätt i det receptet. Sedan skulle den optimala ingrediensen av kandidaterna bestämmas genom beräkningar av näringsvärden samt klimatavtryck. Denna teori är otestad och kräver stora mängder data som i nuläget inte finns i systemet.

6.7.4 Näringssortering

För att uppfylla eventuell brist av något eller några näringsämnen vid menygeneringen skulle en möjlig lösning vara att komplettera menyn genom att lägga till en eller flera ingredienser i ett recept. För att kunna göra det skulle en funktion som sorterar livsmedel efter näringsinnehåll behövas, så att det livsmedel som bäst kompletterar menyn kan läggas till.

6.7.5 Menygenerering med fler parametrar

I avsnitt 6.4 diskuteras ett sätt att ta fram menyer som kombinerar minimering av klimatavtryck och mängden rester. På samma sätt kan parametern hälsa ingå i menygenereringen och på det sättet kan menyer som är bra i fler avseenden tas fram. En vidareutveckling skulle kunna vara att kombinera menygenereringen med receptoptimeringen så att menyerna består av användarens optimerade recept. Recepten skulle kunna anpassas efter menyn, så att de på bästa möjliga sätt bidrar till att de olika näringsbehoven uppfylls. Vidare bör även kraft läggas på att förbättra tidskomplexiteten eftersom den inte är hållbar för en större mängd beräkningar i dagsläget.

6.7.6 Implementering av recepttyper

En kategorisering av recepten utifrån dess typ skulle ge mer flexibla menygenereringar. Det skulle då vara möjligt att söka på mer än bara middagar eller luncher. En möjlig vidareutveckling är alltså att lägga till möjligheten att välja ett antal frukostar, mellanmål, luncher och middagar till menyn, där de olika kategorierna tilldelas olika procentsatser av dagligt intag. Användning av receptkategorier skulle ge mer kompletta menyer.

6.7.7 Dieter

Idag är det många som följer olika dieter, exempelvis i form av LCHF, vegetarisk kost eller glutenfri kost. En möjlig vidareutveckling av MatCoachen är att skapa funktioner för att genenera menyer som följer dessa koster. I Fineli finns livsmedlen kategoriserade efter specialkost, exempelvis glutenfritt, äggfritt eller veganskt. Funktionen skulle med hjälp av dessa kategorier kunna sålla ut recept som inte ingår i en viss diet.

7 Slutsatser

För att motorn ska vara pålitlig är det en förutsättning att recept och menyer med korrekta ingredienser och mått ska tas fram till användaren. Det ställer höga krav på både parsningsprocessen och algoritmerna. Parsningen behöver identifiera rätt ingrediens för att beräkningarna som algoritmerna utför ska kunna grundas på korrekt data. Om fel ingredienser hittas lämnas fel närings- och klimatavtrycksvärden till algoritmerna och ett recept med fel ingrediens, fel näringsinformation och fel klimatavtryck presenteras för användaren. Algoritmerna måste göra beräkningar som ger recept och menyer med närings- och klimatavtrycksvärden som både håller sig innanför de satta gränserna och är anpassade efter unika användare.

Parsningen ger i vissa fall fel ingredienser utan att det noteras av systemet, vilket gör att det tillhörande receptet och all dess information blir felaktigt. I inledningen ställdes frågan: ”Hur goda resultat kan strängmanipulering och reguljära uttryck ge med uppgiften att identifiera rätt ingrediens och mängd i godtyckliga recept från nätet?”. Problemet är lösbart med strängmanipulering som teknik eftersom 299 av 300 rader i stickprovet identifierade ett korrekt livsmedel (se 5.1.1), även om ytterligare 29 av dessa hade mindre problem med enheter och alternativa livsmedel. Antalet rader som bedömdes som felaktiga går att minska avsevärt genom att endast lägga till fler datapunkter i databasen, fler enheter, fler värden för densitet, och så vidare. Strängmanipulering som tillvägagångssätt bör inte lastas för dessa felaktigheter.

Däremot konstateras att parsningen måste matcha en rad helt korrekt för att applikationen ska vara pålitlig, om inte manuella ändringar för felparsade ingredienser ska behöva göras. Därför duger inte parsningsfunktionen för syftet att använda informationen i algoritmuträkningar trots de relativt goda resultaten; parsningen måste alltid identifiera rätt ingrediens och mängd, eller upptäcka själv när den inte gör det, vilket den för tillfället inte gör. I dagsläget finns inte heller alla nödvändiga datapunkter inlagda för alla livsmedel, vilket gör att algoritmerna inte alltid får korrekt information även om livsmedlet är rätt. Om det skulle gå att garantera att alla ingredienser som faktiskt identifieras är korrekt matchade och om rätt datapunkt skulle existera för alla livsmedel skulle det förmodligen gå att säga att parsningsdelen av applikationen var pålitlig, trots att det fortfarande skulle finnas förbättringspotential.

I inledningen beskrevs att rapporten skulle analysera hur goda resultat receptoptimeringen kan ge med avseende på hälsa och miljö. Testerna som utfördes på receptoptimeringen gav bra resultat både i avseende på hälsa och miljö; de optimerade recepten ger i genomsnitt både bättre näringsinnehåll och lägre klimatavtryck än originalrecepten. Däremot har testet som räknar ut näringsvärdet i recepten vissa brister och att recepten blir hälsosamma kan därför inte säkerställas, vilket finns mer beskrivet i 6.2. Vidare har inte smaken på recepten utvärderats, men här går det att anta att smaken inte alltid blir tillfredsställande. Eftersom exempelvis förhållandet mellan vätska och fasta ingredienser efter en receptoptimering kan bli oproportionerligt så kan ett recept bli oätligt eller svårt att tillaga. Andra gånger kan ett optimerat recept vara fullt rimligt. Dessa utfall är dock inte kontrollerade eller stabila. Här finns det rum för förbättringar. Framförallt behövs

någon slags undersökning och kontroll på hur mycket det går att ändra på proportionerna av ingredienserna i recept utan att det blir oätligt eller svårt att laga.

Det beskrevs även i inledningen att rapporten skulle undersöka olika metoder för att generera menygenereringen med både gott resultat och god tidskomplexitet. Menygenereringen är, även med den giriga algoritmen, för långsam för att vara användbar. På grund av just den höga tidskomplexiteten har inte några storskaliga tester genomförts och därför går det inte att dra några slutsatser om hur bra menyer som genereras i det generella fallet. När menygenereringen har testats småskaligt har den gett bra resultat ur både närings- och klimatavtryckssynpunkt, däremot samverkar dessa två parametrarna inte med varandra. En användare bör inte behöva välja om hen vill generera en meny med tanke på näring eller klimatavtryck.

Sammanfattningsvis är inte motorn klar att bygga en pålitlig applikation på ännu. En bra grund har dock lagts och framförallt är problemen väldefinierade. Ett potentiellt fortsatt arbete med systemet har goda förutsättningar med analysen av det här projekt som bas.

Referenser

[1] Elin Röös. *Mat-klimat-listan 1.1*. 2014, s. 10–11. URL: http://pub.epsilon.slu.se/11671/ (hämtad 2017-05-02).

[2] Ola Danielsson. *Äta rätt - både svårt och lätt*. URL: http://ki.se/forskning/ata-ratt-bade-svart-och-latt (hämtad 2017-05-09).

[3] Eva Fröman, Lotta Brinck och Carin Enfors. *Mums miljömat*. URL: http://www.miljomat.se/ (hämtad 2017-04-07).

[4] Kost och Näringsdata AB och Aptimera. *Dietist Net*. URL: http://www.kostdata.se/ (hämtad 2017-04-07).

[5] Viktor Hedefalk och Jimmy Flink. *Kostbevakning*. URL: http://kostbevakningen.se/ (hämtad 2017-04-07).

[6] Okänd. *Labbet*. URL: http://livsmedelsinfo.nu/matlabb.aspx (hämtad 2017-04-27).

[7] Lars Bäckström. *2. Optimering - Linjär programmering*. URL: http://www8.tfe.umu.se/courses/energi/Simulering_och_optimering_av_energisystem/2011/Linear-optimering.pdf (hämtad 2017-04-20).

[8] Daniel A. Spielman och Shang-Hua Teng. *Smoothed Analysis of Algorithms: Why the Simplex Algorithm Usually Takes Polynomial Time*. URL: http://www.di.ens.fr/~vergnaud/algo0910/Simplex.pdf (hämtad 2017-04-25).

[9] Jon Kleinberg and Éva Tardos. *Design Algorithms*. Pearson Education Limited, 2013. ISBN: 9781292023946.

[10] Wolfram MathWorld. *L^2 − Norm*. URL: http://mathworld.wolfram.com/L2-Norm.html (hämtad 2017-04-18).

[11] Nordic Council of Ministers: Köpenhamn. *Nordic Nutrition Recommendations 2012*. 2014. URL: https://www.norden.org/en/theme/former-themes/themes-2016/nordic-nutrition-recommendation/nordic-nutrition-recommendations-2012 (hämtad 2017-04-24).

[12] J. Arthur Harris och Francis G. Benedict. *A Biometric Study of Human Basal Metabolism*. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1091498/pdf/pnas01945-0018.pdf (hämtad 2017-01-31).

[13] Livsmedelsverket. *Energi, kalorier*. URL: https://www.livsmedelsverket.se/livsmedel-och-innehall/naringsamne/energi-kalorier (hämtad 2017-01-31).

[14] Hanna Eneroth, Lena Björck och Åsa Brugård Konde. *Bra livsmedelsval baserat på nordiska näringsrekommendationer 2012*. URL: https://www.livsmedelsverket.se/globalassets/rapporter/2014/2014_livsmedelsverket_19_bra_livsmedelsval.pdf (hämtad 2017-01-31).

[15] Play Framework. *Play Framework*. URL: https://www.playframework.com/ (hämtad 2017-04-07).

[16] Gene Milener och Craig Guyer. *Structured Query Language (SQL)*. URL: https://docs.microsoft.com/en-us/sql/odbc/reference/structured-query-language-sql (hämtad 2017-04-07).

[17] Oracle Corporation. *MySql*. URL: https://www.mysql.com/ (hämtad 2017-04-07).

[18] Rob Bygrave. *Ebean*. URL: http://ebean-orm.github.io/ (hämtad 2017-04-18).

[19] Emil Stenström. *JSON Tagger*. URL: https://json-tagger.com/ (hämtad 2017-04-24).

[20] Jana Straková Milan Straka. *UDPipe*. URL: https://ufal.mff.cuni.cz/udpipe (hämtad 2017-04-24).

[21] Apache. *Apache Commons Math*. URL: https://mvnrepository.com/artifact/org.apache.commons/commons-math3/3.6.1 (hämtad 2017-05-05).

[22] Yasser Ganjisaffar. *crawler4j: Open Source Web Crawler for Java*. URL: https://github.com/yasserg/crawler4j (hämtad 2017-04-18).

[23] Jonathan Hedley. *jsoup: Java HTML Parser*. URL: https://jsoup.org/ (hämtad 2017-04-18).

[24] Vue.js. *Vue.js*. URL: https://vuejs.org/ (hämtad 2017-05-08).

[25] JQuery. *JQuery*. URL: https://jquery.com/ (hämtad 2017-05-08).

[26] Bootstrap. *Bootstrap*. URL: http://getbootstrap.com/ (hämtad 2017-05-08).

[27] Livsmedelsverket. *Näringsämnen*. URL: https://www.livsmedelsverket.se/livsmedel-och-innehall/naringsamne (hämtad 2017-03-28).

[28] Kurera. *Näringsguiden*. URL: http://kurera.se/kategori/naringsguiden/ (hämtad 2017-03-28).

[29] Livsmedelsverket. *Livsmedelsdatabasen*. URL: https://www.livsmedelsverket.se/livsmedelsdatabasen (hämtad 2017-01-31).

[30] EuroFIR. *Food Classification*. URL: http://www.eurofir.eu/about_eurofir/ (hämtad 2017-01-31).

[31] Danish Food Informatics. *LanguaL*. URL: http://www.langual.org/ (hämtad 2017-04-18).

[32] Institutet för hälsa och välfärd. *Fineli är en nationell livsmedelsdatabas*. URL: https://fineli.fi/fineli/sv/tietoa-palvelusta (hämtad 2017-05-12).

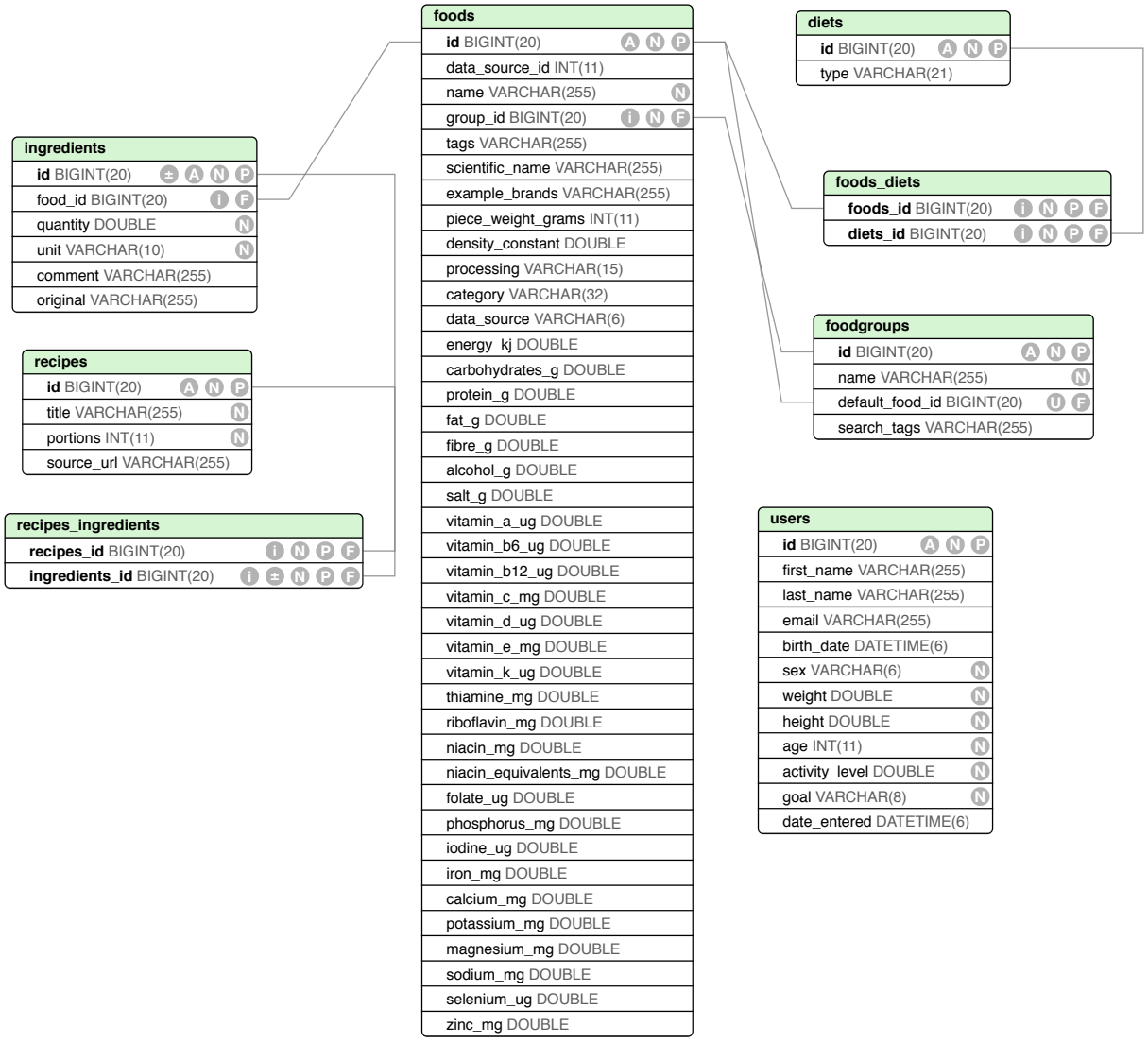
[33] USDA. *USDA Food Composition Databases*. URL: https://ndb.nal.usda.gov/ndb/ (hämtad 2017-05-07).

[34] Henrik Matsson. *Indisk kycklinggryta med curry och grädde*. URL: http://receptfavoriter.se/recept/indisk-kycklinggryta-med-curry-och-graedde.html (hämtad 2017-04-18).

[35] Henrik Matsson. *Receptfavoriter*. URL: https://receptfavoriter.se/ (hämtad 2017-05-12).

[36] WolframAlpha. URL: http://m.wolframalpha.com/input/?i=quartic+fit&x=0&y=0 (hämtad 2017-05-02).

A Diagram över databasen



B Utvärdering av arbetsprocessen

Arbetsprocessen var uppstrukturerad med två möten i veckan, ett i början av veckan där arbetsuppgifter som skulle utföras under veckan delades ut, och ett möte i slutet av veckan där föregående möte följdes upp. Om arbetsuppgifter inte var slutförda kunde detta diskuteras. Varför det ej var gjort och om man behövde hjälp för att ta sig vidare.

Det var lite svårt att dela ut separata arbetsuppgifter under de första fyra veckorna eftersom många möten ägnades åt att diskutera vad projektet skulle innehålla. Att detta tog såpass lång tid berodde på att projektbeskrivningen var väldigt bred och att det fanns många möjliga tillvägagångssätt. Många prioriteringar och överenskommelser behövde göras innan själva arbetet kunde startas.

När arbetet väl kom igång delades gruppen upp i två. Ena halvan började med att implementera recept, livsmedel och användarprofil. Den andra halvan började teoretiskt bygga upp uträkningarna för receptoptimeringen och menygenereringen, samt ta fram en lämplig algoritm för dessa uträkningar. I den teoretiska gruppen hade man kommit fram till ett resultat ganska fort, jämfört med den första gruppen som hade ett mer krävande arbete med att sätta upp systemet. När användarprofilen och livsmedeldatabasen var uppe lades recept in manuellt i receptdatabasen så att beräkningarna för receptoptimeringen och menygenereringen kunde implementeras.

Efter halva projektets gång utformades en första iteration av systemet där målet var att menygenereringen skulle fungera, och att en del av systemet därmed skulle vara fullt fungerande. På detta strukturerades arbetet upp ytterligare. Fler funktionaliteter implementerades vecka för vecka fram tills tre veckor från dess att rapporten skulle in. Ett så kallat kodstopp sattes, för att sedan fokusera på rapporten. Tanken med detta var att se till att få klart rapporten i tid, och använda eventuell överbliven tid till förbättringar i koden och för att utveckla frontenden.

Tanken var att grupperna skulle slagit ihop sitt arbete tidigare men eftersom parsningsprocessen och systemkonstrueringen blev mer omfattande än väntat fick algoritmgruppen testa på en liten mängd recept under en längre tid än planerat. När parsningsprocessen väl var funktionsduglig blev det ändå svårt att använda algoritmerna på en stor mängd recept. Dels på grund av att tidsåtgången växte markant samt att recepten inte blev korrekt parsade till viss del, vilket gjorde uträkningarna felaktiga också.