DM565 Innovation Project

Peter Heilbo Ratgen - perat17
Henrik Schwarz - hschw17
Lucas Olai Jarlkov Olsen - luols17

18th December 2019

Contents

1 Introduction **3**

2 Idea Description **3**

3 Business Model Canvas **3**
3.1 Offering . . . 3
3.2 Customers . . . 4
3.2.1 Customer Segments . . . 4
3.2.2 Customer Relationships . . . 4
3.2.3 Channels . . . 5
3.3 Infrastructure . . . 5
3.3.1 Key Activities . . . 5
3.3.2 Key Resources . . . 6
3.3.3 Key Partners . . . 6
3.4 Finances . . . 6
3.4.1 Cost Structure . . . 6
3.4.2 Revenue Streams . . . 7

4 Prototyping **7**
4.1 System Architecture . . . 7
4.2 Scraping Data . . . 8
4.3 Sanitizing Data . . . 9
4.3.1 Creating the Ingredient Frequency Collection . . . 9
4.3.2 Populating the Ingredient Collection . . . 9
4.4 Getting Prices . . . 10
4.5 User Interface . . . 11
4.5.1 Generating the html . . . 11

A Code Amount **12**

1 Introduction

This report concerns the innovation project in the Formal Languages and Data Processing course. The task of this project is to find an idea for a product involving some type of open data, and then evaluate the this idea as a basis for a startup in a structured manner with the business model canvas. Finally we need to construct a prototype showing the idea in practice.

2 Idea Description

The basis of the idea is to web scrape recipes of different sites, to create a database of various recipes and the ingredients of these recipes. This database of recipes and ingredients will then be paired with the cost of these ingredients included in the recipe. This will enable us to show an estimate of the cost of making the recipe. The cost of the different ingredients can be found using the public product suggestion API made by the retail company Salling. This API references the prices on the `bilkatogo.dk` website. Another resource for current pricing information is from the `etilbudsavisen.dk` API, which shows the current offers of local chains. Prices are also available on some pages that do not have any APIs. Here is another place that we may need to use web scraping. This price information can be used to let the users query recipes based on their specific budgets. Another way for the product to utilize the price information, is that the user can input their current owned ingredients, and get suggestions on recipes of which the user is only missing a few ingredients to be able to make. This can help combat the waste of food.

3 Business Model Canvas

For evaluating our idea the business model canvas will be used. The business model canvas is a template for systematically documenting the business model of an existing business or prospect startup. The business model canvas focuses on four different key areas of a building a sustainable business:

1. Offering
2. Customers
3. Infrastructure
4. Finances

3.1 Offering

The offering is the our value proposition. The value proposition is the services that our product provides. The value proposition is what sets the product apart from the competition. The value which we provide to the customer, is to give the ability to view prices of items that are in a given recipe, this will

also present the total cost of the recipe to the customer. The product will also provide value in enabling the customer to submit their own recipes, to share with other customers. Given that we have different recipes with price data, we can also provide the users with options for a number of recipes given different price points. The problem we solve for the customer is the issue of estimating the cost of a given recipe. Another problem, which we could solve for the customer is the issue of finding a recipe for which the user already has the ingredients (or some subset of ingredients). Solving this problem for the customer can also contribute to less wasted food. It is clear that different services can cater to different customer segments. Eg. more cost-concious consumers may be drawn to the feature of a total cost estimate of a given recipe. The general the value proposition of our product is to provide a service, which gets the job done, and offers convenience for the customer.

3.2 Customers

3.2.1 Customer Segments

At first we must examine who our customers are, in order to more efficiently provide value for them. However ultimately our target is to create value for a mass market audience, since we provide recipes, and most people will at some point search for a recipe. However our core customers are those which contribute recipes, and participate actively in the community around sharing recipes. This would thus equate to people interested in cooking, baking and the like. Another core customer base is the frequently returning customer, which primarily uses the service for looking up cost of recipes, this is the cost-concious customer. The cost-concious customer can be people from most walks of life, it could be a cash-strapped student or the working mom looking to reduce the grocery shopping bill. Returning customers also generate more data, which could be used to suggest recipes, or even better advertisements. Thus there are three overall groups of customers.

- The customer attracted from a search engine
- The contributing, food-enthusiast, which is returning customer
- The returning cost-concious customer

Regarding our three customer segments it would be more than ideal to perform some market research if the people in these three categories are willing and able to use a product like ours. Or if they are already satisfied with the solutions that they have it this point in time.

3.2.2 Customer Relationships

Establishing and maintaining customer relationships is an integral part of retaining returning customers. The customer relationship is also a core part of turning briefly visiting customers into returning customers. However we must

approach the different customer segments in their own way as they are using the site for different purposes. Regarding expectations from customers, the briefly visiting customer does not expect us to establish a relationship with them. They are looking for our content nothing more. However through relevant channels we could entice some small amount of briefly visiting customers to become contributor or returning customers. For returning customers (and contributors) we must continually engage with through relevant channels.

However we are looking to keep the costs of maintaining customer relationships low, as continuously creating engaging content can be costly. Thus we must interact with the customers in terms of an automated service, serving the content they are looking for.

3.2.3 Channels

Customer channels are a crucial part of establishing and maintaining customer relationships. Our product can engage with customers through different channels. For our returning customers we can connect with through mail, so building an e-mail list could be very advantageous. Through mail we can incentivise participation and contribution. For our cost-concious customers we can highlight discounted recipes, or other relevant offers. However the best way to build a great relationship with any customer (including the ones which only visit briefly) is through having the best content. This could be reviews of the different recipes, great photos and other informative content to help the customer engage with the recipe. Thus the goal is to create a product where each type of user can form their own individual relationship with the service, utilizing the service for their specific needs.

We can also generate more reach through targeted online ads. However the primary channel, from which we can acquire new customers is through online search engines.

3.3 Infrastructure

3.3.1 Key Activities

The key activities, are the most important activities for executing our value proposition. To provide value for our customers we examine which key activities our value proposition requires. In our key activities we first and foremost focus on problem solving:

1. To provide value in terms of offering a price estimate on recipes, we first and foremost must establish a system, which handles recipes and the ingredients of these recipes. The ingredients of the different recipes must be linked with prices of products.

2. We must provide a way of presenting the recipes to the user in appealing way.

3. Providing ways for the customer to participate in a community, thus engaging the customer and pushing them to share recipes and thus provide value for other customers.

3.3.2 Key Resources

Key resources are the resources, which our value propositions, distribution channels, customer relationships and revenue streams require. One of our key resources is the data generated by the customers, when they submit recipes. Another key resource is the public API provided by Salling, which aides in producing the prices of individual ingredients. Active users is also a key resource, in that they bring in advertisement revenue. Thus it is important to keep a good relation to customers, such that they come back for more recipes.

3.3.3 Key Partners

A key partner is Salling, which provides pricing data from their open API. This pricing data is hard to source from other places, thus they are an integral partner to our business. A key supplier is the hosting service, which delivers hosting for our website. A hosting service should provide a reliable service, such that we can guarantee availability to our customers. Another key partner is various search engines (although mostly Google), this is of utmost importance, since the vast majority of site visits will come from people searching for a specific recipe. Thus optimizing our service for discoverability (Search Engine Optimization) is very important in acquiring new consumers.

Down the line we could also incorporate prospect key partners, these could be different retail chains sharing prices with us. This could enable us to provide more value to our customers in terms of comparing prices and enabling competition.

3.4 Finances

3.4.1 Cost Structure

Our business focuses on being cost-driven, thus minimizing costs and increasing automation. It is important to reduce cost, eg. in not having a manual review process for the submission of ingredients and recipes, but having this process automated. We can then focus the remaining funds on creating more value for customers, this could be creating pictures associated with different popular recipes.

The cost structure of the product include variable cost like hosting fee, which rise with the amount of customers. There could also be some fixed costs like salaries for content production if that becomes a priority, however this would only be viable after the product the reached a certain scale.

3.4.2 Revenue Streams

Regarding the revenue streams of the product, we can assume the customers are not willing to pay up-front for our services. Since we are catering to cost-concious consumers, and food-enthusiasts, we cannot expect at any point that they are willing to pay out of pocket. We thus have to seek other revenue streams. The most likely way of generating revenue is through advertisements. Advertisements generate revenue, at no cost to the customer. However one could envision a subscription-based model, where the customers are given more premium recipes to choose from, these could be accompanied by instructional videos. This however requires a large paying user-base to be viable. Returning to the current realistic revenue streams, we can also generate a lot of data from returning customers. This data can be sold to third parties, however we must be concious of the impact on our customers if utilizing with revenue stream.

We most also be mindful of rapid growth, this can be a pitfall if the variable costs quickly outgrow the revenue streams.

4 Prototyping

Our goal in creating a prototype of the idea is to showcase the concept of the idea. We have chosen Python to create the product, as we can produce concise code quickly.

4.1 System Architecture

On figure 1 the overall structure of the system is shown. This is a three layer architecture. Here the User Interface(UI) is the top layer. This is where the user interacts with the system. The Domain layer is where the logic is located. The domain layer interacts with the UI. The Domain layer is where we process requests for eg. a specific recipe. The Domain layer then interacts with the database, requesting the right data in the form of queries. The data scraper is a module to the system. The role of data scraper is to populate the database with the correct data. The data scraper also filters data before inserting inserting it into the database.

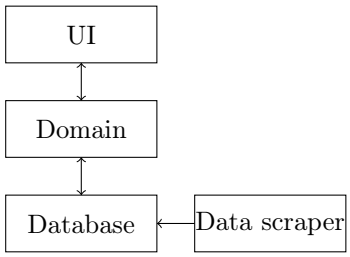

Figure 1: Architecture of the system

Regarding the structure of the database we will need to map the user-submitted ingredients found in recipes to a list of known ingredients. The known list of ingredients, can be used to query price data. Thus we present the following data structure:

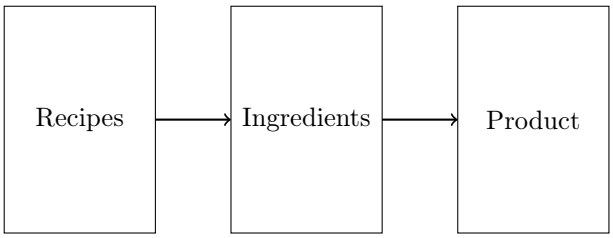

Figure 2: Overview of the data structure

Here we have a list of recipes, the ingredients of these recipes points to a dictionary of ingredients, which all have unique names. There is no recipes, which ingredients are not in the dictionary of approved ingredients. Ingredients can have an alias which points to another ingredient, which is more general. The dictionary of ingredients, each have an array of products that they point to. This achieves a separation of recipes and their prices.

4.2 Scraping Data

We will scraping recipe data from a well-known website called `www.dk-kogebogen.dk`. The recipes from this site are user-made, and are thus filled with all sorts of quirks. As we have using Python we will be using the BeautifulSoup package to help scrap the data from the site. We want to scrape the table containing the ingredients of the recipe, amounts of the ingredients and the units of these amounts. We also want the procedure for scraping the recipe and the title of the recipe. We have identifies that there are in the order of 39027 different recipes numbered in sequential order. As downloading a lot of sites sequentially is quite slow, it makes sense to do this in parallel. Thus the more connections we make to the website we would like to scrape, the faster we can get our results.

```
def execute_get_in_parallel(recipe_id, no_recipes, function):
    urls = create_urls(recipe_id, no_recipes)
    results = []
    thread_count = multiprocessing.cpu_count()
    with ThreadPoolExecutor(max_workers=thread_count) as executor:
        futures = [executor.submit(function, url) for url in urls]
        for result in as_completed(futures):
            results.append(result.result())
    return results
```

With the function `execute_get_in_parallel` we give it the `recipe_id` to start from and the `no_recipes` to scrape. We then pass the name of a function

to be run in parallel. At first a list of urls to crawl is created. Then we take the `cpu_count` of the system to determine the amount threads to spawn. We then create a `ThreadPoolExecutor` with `cpu_count` threads. In line 6 we then submit the `function` and `url` for a thread from the pool to work on. As this happens asynchronously, we have no guarantee for when the executors will be done working. Because of this the executors returns the results in a `Future` object. We can iterate over this list of `Futures` as the executors finish. We then the results of all the functions are returned in a list for further processing.

The function to get ingredients from a url is `get_ingredients(url)`, this function exclusively extracts the ingredients of recipe. It extracts this data from a specific table in the HTML document. This is used make a dictionary of the frequencies of different "ingredients" present in a recipe.

4.3 Sanitizing Data

4.3.1 Creating the Ingredient Frequency Collection

When working with user-submitted data we face a huge issue of varying name to describe the same thing. An example of this is: 'pepper' and 'black pepper', which essentially is the same thing. However our system of course does not know this. The ingredient data also contains parentheses and other special characters, which to not belong. There is also numerous typing errors, and many ingredients have adjectives attached eg. 'large onion', 'cold water', etc. thus we need a way to identify, which ingredients are the correct ones. We decided that the most frequent ingredients used must be of some validity, the same type errors are unlikely to manifest themselves over and over in a relatively limited dataset. At first we decided to filter out all ingredients with special characters with a regular expression. `'.*(\(|-|/|:|\.).*'`, thus ingredients containing these are not considered. When inserting the frequencies of the ingredients into the `ingredient_frequency` collection in the database, we need to count how many times the ingredient occurs in a list. However this list contains *every* ingredient from every recipe(apart from the ones containing special characters). To optimize the running time of this, when we have created a dictionary with the name of a ingredient and its frequency, we insert the name of the ingredient into a dictionary. Here we remember that it is much faster to lookup a key in a dictionary as opposed to iterating through an entire list. Thus as we go through item, we have an effective way of not checking for the frequency of the same ingredient over and over. This reduced the running time of creating the ingredient frequency collection from above an hour(did not finish), to about 25 minutes.

4.3.2 Populating the Ingredient Collection

As we have obtained the frequencies of all appropriate ingredients we need to insert these into the database as proper ingredients. We do this in the `populate_ingredient_whitelist()` function. To avoid to many "exotic" ingredient names in our collection of ingredients, we set a minimum frequency of

greater than 10 occurrences. We determined that this threshold was adequate, in that we could reasonably expect at least some recipes would contain these 1994 different common ingredients. However these ingredients are not distinct ingredients. That is why we have aliases, which point from one ingredient to another. We determine if one ingredient is an alias of another by, iterating through the frequency collection, taking the most frequent ingredients first (the higher frequency the more we can trust the integrity of the ingredient) and running a search with the regular regular expression

```
'(.* {0}$|^{0}.*)'.format(item['item'])
```

This regular expression matches the words beginning with 'name' and the word ending with 'name' preceding a space. This does catches adjectives attached to an ingredient. Then the word which matches the regular expression, is inserted with an alias to the more frequent ingredient. This is not a perfect approach, but it gets the job done. The database is populated continuously throughout the iterations.

4.4 Getting Prices

We obtain prices from the aforementioned `api.sallinggroup.com` API. Here we use their product suggestion API, we send query in the form of a string. The API then sends back relevant product suggestions, in a JSON file: thus we receive data in the form of:

```
{
  "title": "Naturmaelk Bio Minimaelk 0,5 1 L",
  "id": "93008500001",
  "prod_id": "19680",
  "price": 11.95,
  "description": "",
  "link": "https://www.bilkatogo.dk/s?query=19680",
  "img": "https://image.prod.iposeninfra.com/bilkaimg.
   php?pid=19680&imgType=jpeg"
}
```

a list of suggestions containing the title, id, prod_id, a price, for some items there is a description, and finally links to the site where the item can be bought, and a link to an image of the product. We have a Product object where we store the data about each individual product. We hold the EAN, name, amount, unit, most recent price, and the price history. Whenever we query a product, we will check in our database to see if the observed price is different from the last price we observed, in which case we will update the price add it to our price history. Individual ingredients(without aliases) are also connected to different products in their `product_list`. Thus the product list is consulted each time an ingredient is queried.

4.5 User Interface

4.5.1 Generating the html

Jinja2 For our user interface we have used html, which makes sense when making a webapp. To generate our html templates we have used a templating engine called Jinja2 which our app engine flask supports. Jinja2 is widely used for python web framework, because it is fast, secure and easy to use.

The language can use control expressions like if statements and for loops. It also use of blocks and macros. Blocks are used for template inheritance which allows us to create reusable templates. The templates are created as html files where we add the control structures using Jinja2 expressions. We have defined a base template that imports css and scripts that are used by all the templates. Using blocks other templates can add to the base template. In our case the base template is 'templates/main/base.html'. We then reuse that template to create other templates and avoid a lot of duplicate html code.

jQuery and Ajax jQuery is a javascript framework that enables easy HTML DOM, event handling, css animation and ajax requests. It treats html documents like a three structure, which lets us easily add and remove html elements.

Ajax is asynchronous javascript and XML which can be used to make asynchronous web applications. We use it primarily to query data on the back-end and then show it using jQuery on the front-end.

Views Our webapp has to primary views which are:

1. /recipes/
2. /ingredients/

Recipes The recipes view shows all the recipes that are scraped and collected. The recipes can be searched by name by writing in the search input and then pressing enter. We then use jQuery and ajax to query a endpoint that returns all the names that match a regular expression. The recipes can then be access individually using links from the list.

We can also create a new list using the 'lav ny' button which redirects to a view with a form. The form has a field called 'Ingredients' where if we write something of four characters length and press down we get a list of viable ingredients. We then add a amount and a unit and press 'Add ingredients' to add it to the recipe. When filled out we can press 'send' and send the data to our back-end where it is validated and added to the database.

Ingredients Ingredients has a list view just like recipes where all the ingredients are listed. The ingredients can be searched just like recipes and there are likes for individual links for every ingredient.

When we access the individual ingredients we can see the alias for the ingredient and the recipes the ingredient is part of.

A Code Amount

```
. ingredient_product_suggestions.py 43
. mongoenv.py 1
. run.py 6
. scrape_recipe_ingredients.py 199
. source_line_count.py 28
./reciprice/templates/main ingredients.html 78
./reciprice/templates/main ingredient.html 31
./reciprice/templates/main recipes.html 78
./reciprice/templates/main index.html 13
./reciprice/templates/main recipe.html 54
./reciprice/templates/main create_recipe.html 163
./reciprice/templates/main base.html 61
./reciprice extentions.py 5
./reciprice main.py 191
./reciprice salling.py 26
./reciprice __init__.py 20
./reciprice models.py 170
./reciprice settings.py 3
1170
```

The total lines of code is 1170.