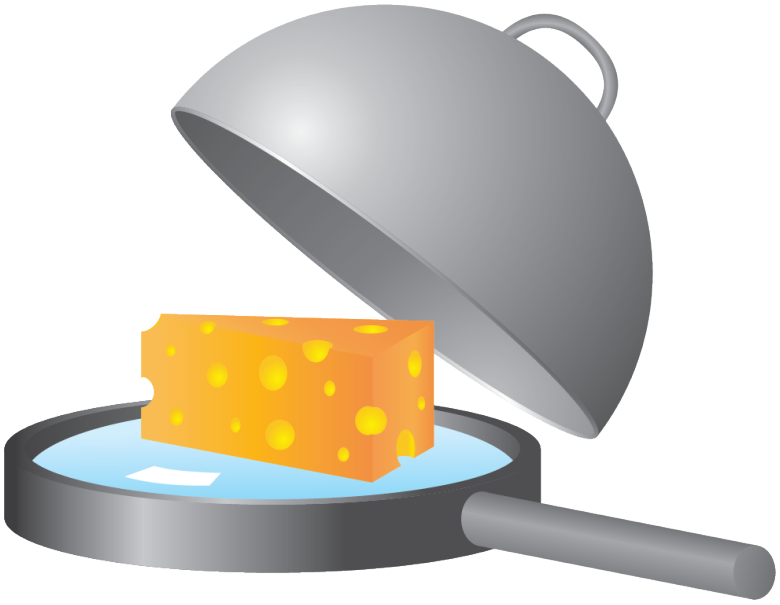
Aalborg University

Dishcover

- An Interactive Mobile Cookbook



AALBORG UNIVERSITY
STUDENT REPORT

Department of Computer Science
Software Engineering
Selma Lagerløfs Vej 300
Telephone +45 9940 9940
+45 9940 9798
http://www.cs.aau.dk

Title:
Dishcover

Subject:
Mobile Systems

Project period:
P8, Spring semester 2014

Project group:
SW805F14

Attendees:
Jacob K. Wortmann
Jesper Riemer Andersen
Nicklas Andersen
Sam Sepstrup Olesen
Simon Reedtz Olesen

Supervisor:
Ramin Sadre

Finished: 28-05-2014

Number of pages: 88

Appendix pages: 8

Synopsis:

The focus of the project is making an application that gives the user quick access to cooking recipes.
It is possible to search for recipes by their ingredients as well as their name and description. While searching for recipes by ingredients, a word cloud aids the user with suggestions for ingredients that could be included in the search.
The main motivation behind the application is that the users should be able to easily and quickly find recipes that include certain ingredients.
The user is able to login using their Google+ profile enabling them to favourite recipes, which allows them to quickly access the recipes again.

Signatures

Jacob K. Wortmann

Jesper Riemer Andersen

Nicklas Andersen

Sam Sepstrup Olesen

Simon Reedtz Olesen

Preface

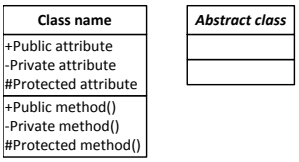
This project was written as a semester project by group SW805F14 - Software students from the Department of Computer Science at Aalborg University in the Spring of 2014. The report documents the implementation of the Dishcover application. The application is developed for the Android platform. The reader is expected to be familiar with Java, UML, and the Android platform. We have included our knowledge from all our previous semesters.

Database access:

URL: **http://figz.dk/phpmyadmin**
Username: **readonly**
Password: **readonly**

When reading the report, there are a few things the reader should be aware of:

- When a reference to a source of a section or paragraph is given, the number of the source is written inside square brackets [ ]. The number is a reference to the bibliography list on page 87.
- When "we/us/our" is mentioned in the report, it is a referral to the authors of the report
- In class diagrams a minus symbol denotes a private attribute/method, a plus symbol denotes a public attribute/method, and a sharp symbol denotes a protected attribute/method. Italic class names are abstract classes.



We would like to thank our supervisor Ramin Sadre for the feedback he has given throughout the project.

Development Environment

To create the Dishcover application and the server service we have used the following technologies.

Application	
Java	1.7
Android Studio	0.5.8
Android SDK	19
Gradle	0.9
Java	1.7

Server	
Apache	2.2.14
PHP	5.3.2
PHPunit	3.6.10
MariaDB	5.5.37

Contents

Analysis *1*

One in every five people in the world today own a smartphone[20], and their popularity among consumers does not seem to decline[21]. Mobile devices allow for easy access to the Internet at all times, which means easier access to a vast amount of information and different kinds of media outside of your home, this includes music, video, and books.

The online media is becoming a good alternative to physical media since it is a lot easier to search for it on your phone or computer which you always have near you. For example it is a lot easier to buy or rent an e-book online with a few clicks on your device than having to go to the library and borrow it. This means that in some cases books are becoming obsolete, among those are cookbooks. A physical cookbook is not going to be replaced purely by online recipes but the online recipes allow people to easily and fast search for recipes without having to borrow or buy a cookbook.

Cookbooks provide a static set of recipes, whereas a mobile device can provide you with a vast amount of recipes. Depending on the size of the device/cookbook, it might be easier to read from a mobile device and more convenient to move it around in the kitchen. A mobile cookbook can also provide a way to search for recipes and easily bookmark recipes you like.

A mobile cookbook provides the option to search for recipes when you want it. You can search for recipes while you are in the supermarket in order to find out which ingredients you need for a specific dish. This means you do not need to plan ahead of your grocery shopping as much. Thus mobile devices allow for a way to assist you in impulse visits to the supermarket.

1.1 Existing Solutions

During the early stages of the project we began searching for existing solutions to draw inspiration to the design and implementation phases. All members of the group have previously had experience using different recipe sites on the Internet, and we think that the existing solutions lack functionality. In this section we review and discuss existing solutions and draw some good practices from these.

Supercook

Supercook[10] is a website with focus on searching for recipes based on their ingredients, however you can also search for recipes using free-text.

When searching using ingredients the user is limited to enter a set of ingredients which is defined by Supercook. When adding ingredients to a search the user is aided by autocompletion and a word cloud with ingredients.

The word cloud changes according to the type of ingredients you have entered, e.g. if you enter “vanilla” the word cloud will typically change to contain ingredients related to desserts. It is possible to apply restrictions to your search, e.g. “I don’t eat meat” and all recipes with meat will be excluded. A search on

Supercook will result in a prioritised list of links to recipes from several online cookbooks. A screenshot of the website is shown in Figure 1.1.

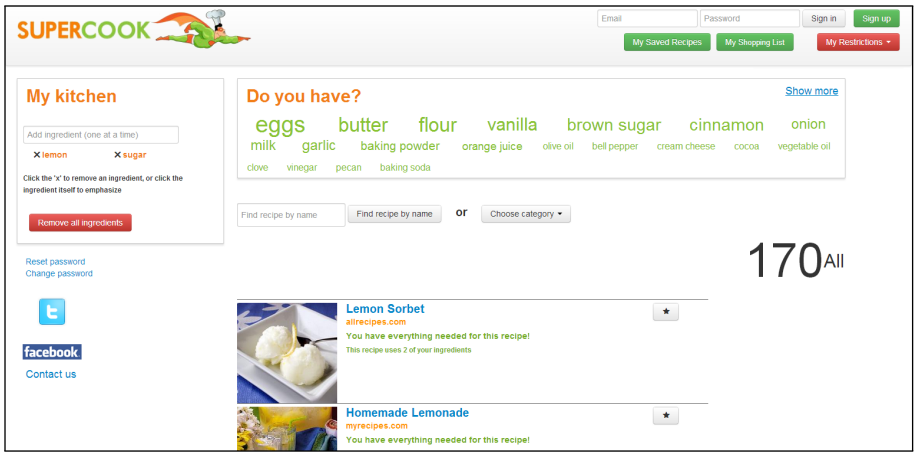


Figure 1.1: The interface of Supercook.

By evaluating the search results of Supercook, we have deduced what we believe are the precedence function used to generate the search results:

1. Remove recipes without any matching ingredients.
2. Sort by most matching ingredients.
3. Sort by least missing ingredients.

Notice that the result is primarily sorted by the least missing ingredients since it is the latter sort.

We have also noticed some quirks in the way it counts matching ingredients. It seems like the search algorithm counts the number of occurrences of the entered ingredients in the recipes. When entering “bell pepper” and “olive oil”, recipes which contain both red, green, and yellow pepper are prioritised higher than recipes which actually contains both of the entered ingredients. It also appears as if ingredients in the recipes are mapped to the ingredients defined by Supercook. An example of that is that “flour” appear to be mapped to “flour”, “wheat”, “oat”, etc. This also appears to have the shortcoming that recipes containing e.g. “wheat flour” are prioritised as if they have two matching ingredient when the user entered “flour”, since Supercook detects both “wheat” and “flour”.

When making a search containing “eggs”, “garlic”, “ground beef”, and “onion”, we get a result which showcases a drawback of the sorting done by Supercook. The first 94 recipes are simple recipes using only few of the entered ingredients, including 24 recipes on how to boil an egg. Recipe number 95 is a recipe for meatballs and is using all entered ingredients but it also needs “bread crumbs”. The reason why the first 94 recipes are prioritised highest is because they do not need any other ingredients than the ingredients entered.

Allthecooks

Allthecooks is one of the most downloaded [1] Android applications that is related to the search: "recipe". The application design follows the Android guidelines[6], i.e. the application follows the design patterns recommended by Google. By following the guidelines an Android user can more easily become familiar with the application, since the user previously has had experience with other applications which also follows the guidelines. The detailed display of a recipe is especially well implemented, the user has everything on a single page, and can easily see the needed ingredients and the required steps to cook the recipe. The user can also, by the press of a button, toggle between different measuring units, or add the selected ingredient to a shopping list. Screenshots of the application are shown in Figure 1.2, 1.3, 1.4, and 1.5.

The recipes of Allthecooks are user created, meaning they are created and uploaded by users. The photos are also provided by users, this can be dangerous if the photos are not checked for content not suitable for the application's audience. An advantage of user created recipes is that the applications content is growing with the audience of the application. The search is free-text based, which means that opposite the Supercook web application it is hard to find recipes based on ingredients. You are able to apply filters to your search to remove recipes that contain certain ingredients.

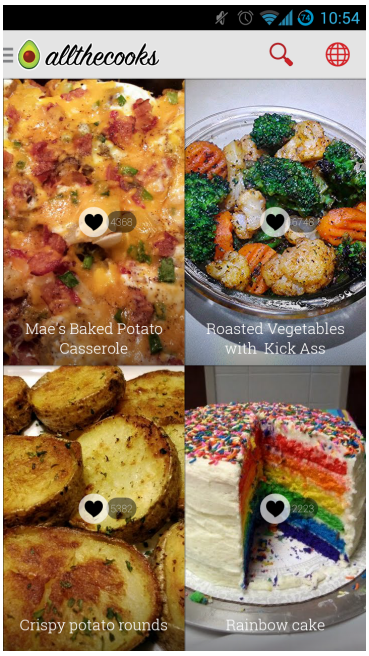


Figure 1.2: Menu of Allthecooks.

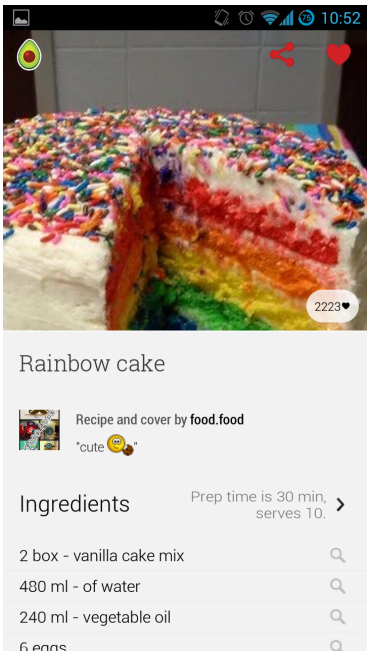


Figure 1.3: Detail display of a recipe.

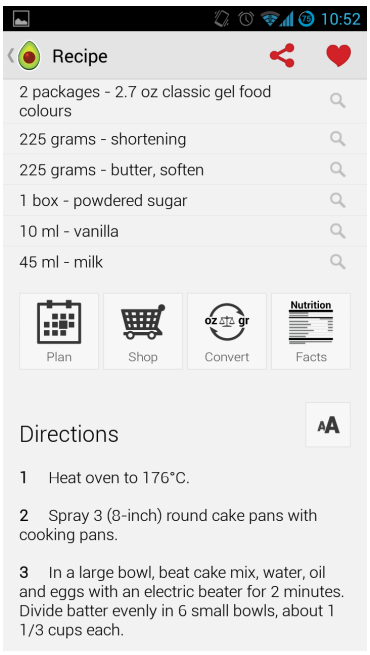


Figure 1.4: Buttons for different features.

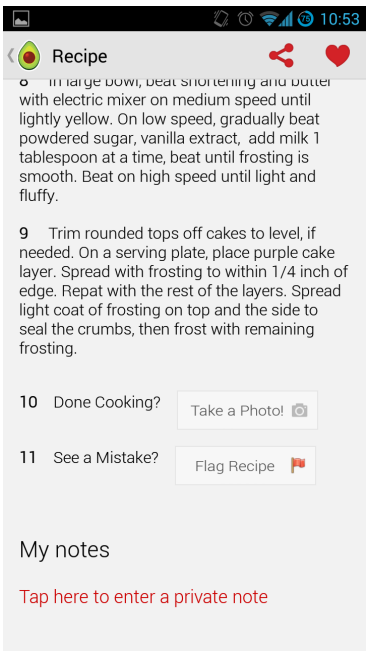


Figure 1.5: Directions for the recipe.

BigOven

BigOven is also one of the most downloaded [2] Android applications that is related to "recipe". The application's navigation and design does not follow the Android guidelines[6] and therefore the application's navigation can be quite confusing to an Android user.

The application main page is cluttered and presents the user with many functionalities, see Figure 1.6. It is possible to search both by ingredients and free-text. The search by ingredients is limited to only three ingredients and the algorithm finds only the recipes where they all are included. The search by ingredient is based on free-text, meaning the user does not have the aid of autocompletion, like with Supercook. If the user inputs three ingredients that have no real relation like: "beef", "cake-mix", "salmon", the search by ingredients would find no results, the search by ingredients excludes everything that does not contain all three ingredients. Like in the application Allthecooks, the user can add the different ingredients to a shopping list, save the recipe to favourites, and alternate between the metric and imperial system.

A feature that is unique to the BigOven application is the menu-cards, see Figure 1.7.

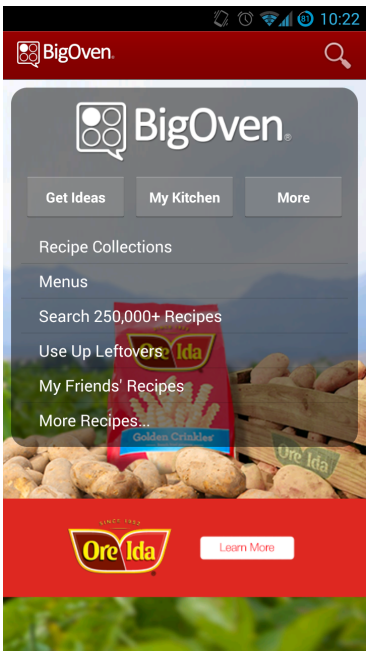


Figure 1.6: The main page of BigOven.

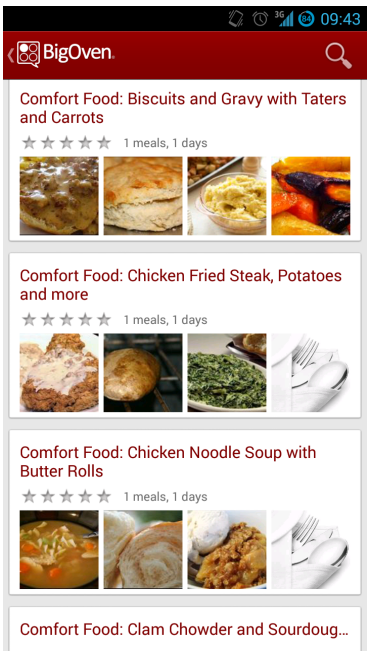


Figure 1.7: Menu Cards from BigOven.

It is a small collection of different recipes that together form a meal, e.g. "steak", "fries", and "green beans". Each meal is bound to a day which means that the user can create a menu-card that can contain a meal for each day of the week or more.

The recipes of BigOven is also, like Allthecooks, user created which provides the same risks and benefits as explained in the previous section. The BigOven application is free but you can buy Pro-features that exclude advertisers from the application and unlocks more functionality in the application.

Comparison

These three applications have been chosen for comparison since they vary in focus, design, navigation, and features. Allthecooks focus on design and navigation, where as BigOven has focus on features. Supercook is a bit different since it is not an Android application but it was included in the comparison since it had a unique set of features, such as the word cloud, and search by ingredients. Allthecooks and BigOven share a lot of features, they both have free-text search, shopping list, and saving of recipes. Though BigOven has some features that are unique, such as search by ingredients, and menu-cards. A noticeable thing about BigOven is that the experience of using the application could be improved significantly, both in terms of its design and performance.

BigOven and Supercook is both able to search by ingredients, but there is a big difference in the two searches. In the BigOven application you are only able to enter three ingredients. The search finds all the recipes that include all three ingredients. BigOven also uses free-text as an input method, whereas Supercook aids the user with autocompletion. In Supercook the user is also

able to enter as many ingredients as necessary. A comparison between the existing solutions can be seen in Table 1.1.

	Supercook	**Allthecooks**	**BigOven**
Platform	Website	Android/Website	Android/Website
Cost	Free	Free	Free & Paid
Recipe search	Yes	Yes	Yes
Ingredient search	Yes	No	Yes
Recipe origin	Web crawl	User created	User created
Shopping list	Yes	Yes	Yes
Menu planner	No	Yes	Yes
Saving recipes	Yes	Yes	Yes
Ingredient filter	Yes	Yes	Yes
Sharing of shopping list	No	E-mail/SMS	E-mail
Sharing recipe	No	App/SMS/E-mail	App/E-mail
Needs Internet	Yes	Yes/No	Yes
Needs login	No	No	No

Table 1.1: Application comparison.

The user does not need to be signed in to search for recipes in any of the applications. In BigOven the user needs to be signed in to save recipes to favourites and adding items to the shopping list. In Allthecooks the user does not need to be signed in to save recipes to favourites, but the user still needs to be signed in to access the shopping list. In all applications the mobile device needs a Internet connection to search and access the recipes, they are not stored locally on the device, though in the Allthecooks application the user are able to view its favourites and shopping list without a Internet connection.

1.2 Obtaining Recipes

An application for finding recipes is not of much value without a sizeable collection of recipes. It would be a major overhead for our project if we had to create all recipes available in the system. This includes a title, description, instructions, and pictures, for each recipe.

If we want to make recipes for the system, we would undoubtedly be inspired by existing recipes. We decided to contact two lawyers to help clarify the Danish copyright law regarding recipes. Their full responses can be seen in Appendix A. It became clear that the law is quite vague about this subject. If we change some ingredients in an existing recipe we can legally call it our own.

> *Recipes - or culinary works if you want - are covered by copyright (in accordance with the law's § 1), as they are a kind of non-fiction works in print.*
>
> — Erik Frodelund, *Familieadvokaten.dk*

> *...are you using a published recipe to develop your very own recipe - for example with more ingredients and a little different quantities - you have created a new culinary work that you are free to publish in for example a book.*
>
> — Erik Frodelund, *Familieadvokaten.dk*

We are also allowed to use recipes which are considered to be "standard-versions". A "standard-version" recipe is vaguely defined as a recipe which is "generally known".

> *As long as you stick to the recipes' standard-content and approach you are not violating any rights.*
>
> — Jørgen Lindhardt Steffesen, *Startvækst.dk*

The obvious disadvantages of creating a recipe based on copyrighted material is that we need to write the description, instructions, and obtain pictures of the resulting dish ourselves.

It is also possible to obtain recipes which are licensed by a *Free Cultural Works*[5] license. *Free Cultural Works* are works which can be used for free, copied, and modified, for any purpose. This includes all works covered by the Creative Commons[3] licenses that allow commercial use and modification of the material. The following websites contain recipes of which many are licensed by *Free Cultural Works* licenses:

- `http://en.wikibooks.org/wiki/Cookbook:Recipes`
- `http://www.nibbledish.com/`
- `http://www.opensourcefood.com/`

No matter how the recipes are obtained, we are required to adapt every recipe to fit our data structure. While some content can be crawled from the sources, different formatting between the content creators makes it hard, if not impossible, to automate the adaptation of the content to our data structure.

1.3 Internationalisation

We want our system to be internationalised such that it is ready for localisation.

Apart from having to provide the ability to add multiple languages, internationalisation also cause other difficulties for some cultures:

Reading direction While most languages are written left-to-right top-to-bottom, some languages, like Japanese, are written top-to-bottom right-to-left.

Special characters Some languages require characters which are not present in ASCII, e.g. "æøå" in Danish.

Text length Text may be significantly longer when translated to other languages. An example is text displaying how many times an item has been viewed, e.g. "5 views". Translated to Italian that is written "5 visualizzazioni"[22].

Number formatting While most of the world is using "dot" as the decimal mark, many cultures are using other decimal marks, like "comma".

Units of measure Most of the world is using metric(SI) units for measures; however, USA, Liberia, and Burma have not yet adopted the metric system[13].

Prohibited foods Some types of food may be prohibited or tabooed in certain cultures and religions.

1.4 Problem Definition

Finding recipes for cooking online has become easy with the vast amount of sites that offer them. This has also extended to the mobile platform, allowing the users to quickly find a recipe wherever they are. It is easy to type in the name of the recipe you want, and it tells you what ingredients you need and a step by step guide on how to cook the dish.

After a small preview of the already existing solutions, of both web and android applications, we believe that we can create an application with improved features, such as a better search algorithm, easy navigation, and an elegant design.

The user does not always know exactly what dish they want to cook but instead know which ingredients they have or what ingredients they want to use, so we like the idea of being able to search for recipes by ingredients but we will not exclude the possibility to also search directly for recipes.

Being able to search for recipes based on ingredients the user already has, allows them to customise their search and receive a larger selection of dishes. It allows them to search based on items they already have in their kitchen, or based on items they know are about to go bad and they need to use. The user could also be in a situation where they see something on sale at the supermarket and quickly need to find out what dishes they can make using this ingredient.

With these problems in mind we have come up with the following problem statement for our project:

> How can we take advantage of the mobile platform, in order to provide the user with relevant recipes based on specific ingredients, and taking any user defined restrictions, like allergies, into consideration.

1.5 Requirements

Based on the research of existing solutions and our problem definition, we have come up with a list of requirements for our application. This list can be seen in this section.

1. **Android**
   The application must be able to run on Android version 4.3 and newer.
2. **Search by ingredients**
   The user must be able to search for recipes containing specific ingredients defined by the user.

3. **Free-text search**
   The user must be able to search for a recipe by its title.

4. **Search filters**
   The different search methods must support filtering for users that might be allergic to certain ingredients.

5. **Favourite recipes**
   The user must be able to save their favourite recipes. They must also be able to remove recipes from their favourite list.

6. **Shopping list**
   The user must be able to add and remove ingredients from a shopping list directly from the recipe. They must also be able to add and remove ingredients directly from the shopping list as well.

7. **Sharing**
   The user must be able to share recipes and shopping lists with other users using Facebook, Google+, E-mail, and SMS.

8. **Persistency**
   The system must be able to keep the user's shopping list and favourite recipes persistent.

9. **Unit conversion**
   The application must support unit conversion between the imperial system and the metric system.

10. **Additional languages**
    The system must be ready for internationalisation.

11. **No login required**
    The user should not be required to log in, to use the application. Though the user might not be able to use certain features if not logged in.

Design

2

This section describes how we have used user stories as basis for our navigation design. The section also shortly describes our system architecture and shows our early and later stages prototypes.

The reader is expected to be familiar with standard Android components. Some components are briefly explained here:

Activity An activity is a single focused thing that the user can do. You can also say that it is a window which is either full-screen or floating [15].

Fragment A fragment is nested in an activity which means its lifecycle is tied to its parent activity. Fragments can be used to build a multi-pane user interface[16].

Action Bar The action bar is a window feature that identifies your location in the application, and provides user actions and navigation modes. It is located at the top of the screen[14].

However, we are also referring to activities and fragments as pages, this is because it is irrelevant to the context what kind of implementation is used.

2.1 Stories

In the following section we create user stories as the basis for finding how the users navigate through our application. The stories are made up and they are trying to cover all the key navigation paths in the application.

Julie

Julie is at the supermarket, she does not know what to make for dinner. She sees that minced beef is on sale, but she can not think of any recipes that include minced beef. She opens the application Dishcover. She is presented with a page that says "Click above to search for ingredients". She clicks the button, and a view of categories appears at the bottom of the screen. She clicks the category called "Meat", and a word cloud appears with many different kinds of meat. In the word cloud she finds minced beef and clicks it. Now she presses the back button and the application performs a search which presents Julie with recipes that uses minced beef. The top results include recipes such as Spaghetti Bolognese and Lasagne. Having not thought about this before, Julie wants to cook Spaghetti Bolognese for dinner. She opens the recipe and can now see all the ingredients she need for this recipe. Remembering which of the ingredients she has at home, she then adds the ingredients she needs to her shopping list. Then she favourites the recipe to easily find it later. Julie opens her shopping list and continues shopping.

Bob

Bob is at home and does not know what he want to make for dinner. He looks in the fridge because there might be something to use. In the fridge he finds some chicken which has a best before date set to today. To get some inspiration he opens the application and types in chicken and a few other ingredients he has in the kitchen. The application provides him with several recipes including chicken, he decides he wants Chicken Tikka Masala. Based on the ingredients he typed in before he discovers that he is missing paprika. He adds paprika to his shopping list and favourites the recipe. His wife Sofie has not arrived at home yet, so instead of going to the supermarket himself, he decides to share his shopping list with Sofie.

Alice

Alice is at home, earlier she found a Lasagne recipe and she has already bought all the ingredients for it. She opens the application Dishcover, goes to the favourites page, and opens her previously favourited recipe for Lasagne. In the recipe she scrolls down to the instructions and follows these in order to make her dinner.

2.2 System Architecture

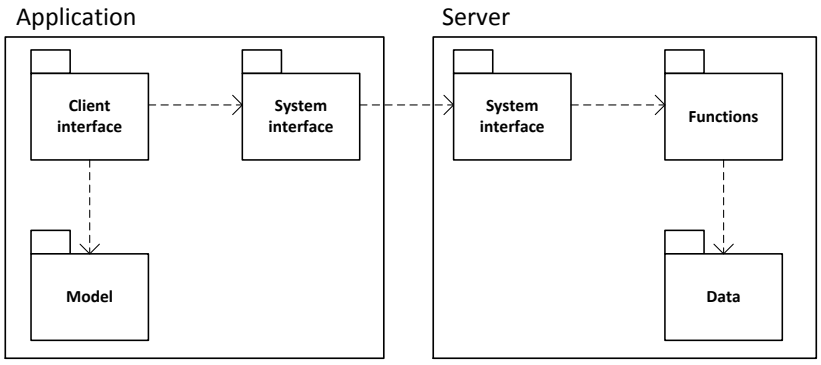


Figure 2.1: System architecture.

Figure 2.1 shows the architecture of the entire system. The mobile application has a client interface, for example when a user wants to view a recipe, a request is made through the application's system interface which then receives the recipe data from the server. The recipe data is mapped to the model which can easily be displayed on the device's screen. The application does not have a Functions component because the functionalities are provided by the server.

The server handles requests from clients with its function component. The function component gets the requested data from the database and is then serialised to a JavaScript Object Notation (JSON) formatted object, so the application later can unsearialise it.

2.3 Prototypes

This section shows the prototypes that we have made and touches lightly on the application's functionality. The section includes both early and late prototypes to show how the design has progressed and changed.

Figure 2.2: The navigation drawer.

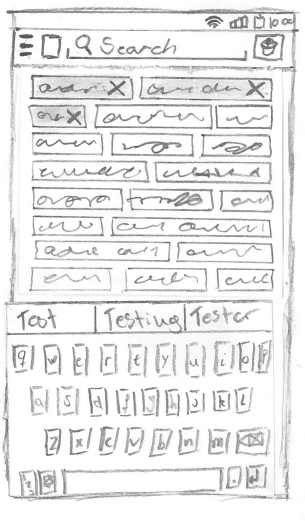


Figure 2.3: Ingredient search with text.

The first thing the user is shown when opening the application is the navigation drawer, as seen on Figure 2.2. This is a standard component in Android which is used to navigate between different pages[7]. We felt that it was a good way to make the views less cluttered, collecting all navigation options in an easy accessible component. An alternative to the navigation drawer could have been to use fixed tabs, however this is only used if there is a limited number of up to three top-level views which we do not have[6].

The application allows the user to search for recipes both based on the title of the recipe and based on ingredients that the user inputs. Searching for recipes based on the title is simple and straight forward, the user inputs a partial or full title of the recipe they want and the application shows suggestions based on this.

Searching based on ingredients is more complicated. The user might have to type in multiple ingredients and we want to make it easy for them to do so, therefore we have implemented two different ways of aiding the user in entering ingredients, either by text or by tiles.

Figure 2.3 shows the screen layout when entering ingredients with text. The user clicks the search field at the top of the page and a keyboard and a word cloud pops up. Now the user can either type in the ingredient they want by use of the keyboard, or if the word appears in the word cloud they can simply click it. The idea of the word cloud is that it provides suggestions based on the ingredients the user has already put in, allowing the user to perform faster

searches as they would not need to type so much but instead be able to click directly from the word cloud.

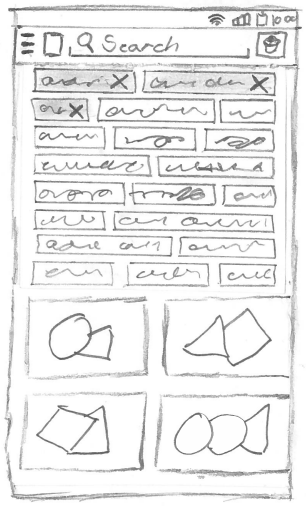


Figure 2.4: Ingredient search with tile selection.

Figure 2.5: Recipe browsing with a word cloud.

The same idea applies when entering ingredients using tiles, but instead of a keyboard a small tile bar pops up. This bar contains the different categories, and the user can click on one of these in order to go deeper down into a category, as the user navigates deeper the word cloud will update with words associated with the category they are browsing. The tile navigation can be seen on Figure 2.4. When the user is done with inputting their ingredients they press the back button to close the keyboard or the tiles and can browse the suggested recipes. Figure 2.5, Figure 2.6, and Figure 2.7 are three different versions of what it will look like when the user does so.

Figure 2.5 shows the same page without the keyboard, giving the user access to browse the suggested recipes while still having access to the word cloud allowing them to remove or add new ingredients on the fly, however the drawback is a very small view of the suggested recipes.

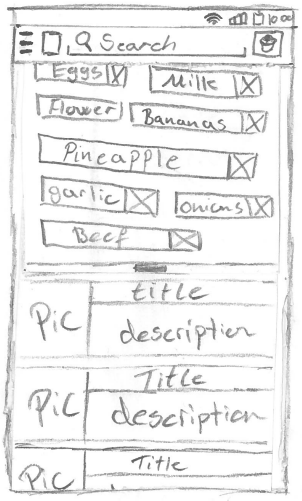


Figure 2.6: Recipe browse with static ingredients.

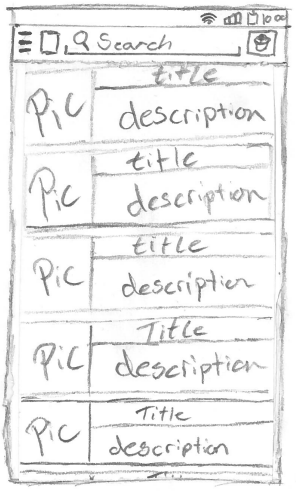


Figure 2.7: Recipe browse without ingredients.

Figure 2.6 shows the page without the keyboard or word cloud but only the selected ingredients, allowing the user to remove items on the fly, again the drawback is a rather small view for the suggested recipes.

Figure 2.7 only shows the suggested recipes, allowing the user to see more recipes but having to click the search field again in order to edit their ingredients.

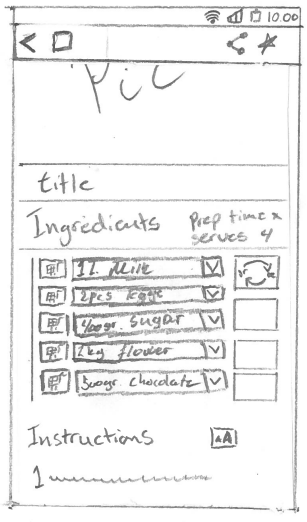


Figure 2.8: First layout for the recipe view.

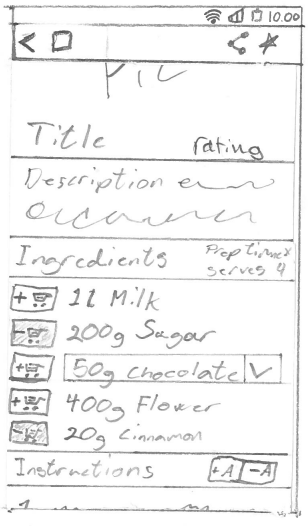


Figure 2.9: Redesigned recipe layout.

When the user finds a suggested recipe they want to look at, they press it and they are taken to a new page. The two variants of this page is shown on Figure 2.8 and Figure 2.9. The three vertical lines in the top left corner have been replaced with a back arrow, this indicates two things; the navigation drawer is not available on this page and the back arrow indicates that if you click it you will be taken back to the previous page which is the list of suggested recipes.

In the top right corner there are two new icons; the first button is the share button, allowing the user to share the recipe with their friends and family. The second button is the favourite button, the user can click this to save the recipe under their favourites if they click it again they remove the favourite.

Figure 2.8 shows the first design, under the action bar there is a picture of the dish and just under that the title. This is followed by a list of ingredients and the instructions for making the dish. Next to "Ingredients" the user can see the time it takes to prepare the dish, and how many people the dish will serve.

To the left each ingredient there is a button with a shopping cart icon, the user can press this in order to add this ingredient to their shopping list. If the user presses the same icon again, the ingredient is removed from the shopping list. By default the application assumes that the user does not have any of the ingredients, so the user has to choose which ingredients they need to add to the shopping list.

Each of the ingredients have a box around them and a down arrow, this is to indicate a drop down menu. These will appear when an ingredient is exchangeable according to the recipe, meaning the user can use something else similar and still use the recipe.

To the right of the ingredients there is a button with a circle on it. If the user presses this button all the quantities in the recipe will be converted between Imperial and Metric and visa versa. Under the list of ingredients there is a list of instructions telling the user how to cook the dish. The button labelled "aA" next to instructions can be pressed and different font sizes can be chosen.

We redesigned this layout and the result can be seen on Figure 2.9. Most of the changes are small changes to previous designs, for example the location of the title. It has now been moved up onto the picture, together with a rating for the recipe. There is now a small description under the recipe describing the dish. The shopping carts now have a small "+" or a small "-" to indicate if the user will add or delete the ingredient when they press it. We kept the drop down menu for ingredients to indicate that an ingredient is exchangeable. Instead of the "aA" icon that the user should press, there are now two buttons, one to make the font larger and one to make it smaller.

One of the larger redesigns we did, was that we removed the buttons to the right of the ingredients as we decided to add their functionality elsewhere and therefore have more room for the ingredient names.

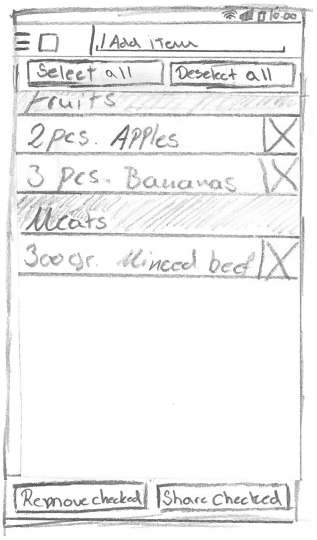


Figure 2.10: First layout for the shopping list.

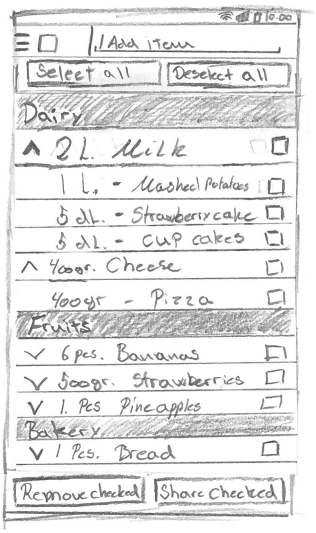


Figure 2.11: Redesigned layout for shopping list.

The user can always access their shopping list through the navigation drawer. Figure 2.10 and Figure 2.11 shows the early design and redesign of the shopping list. The user is able to add ingredients directly in the shopping list by clicking the "Add item" field at the top. They are also able to quickly select or deselect all items in the list by clicking the "Select all" or "Deselect all" button under the action bar. This is useful if the user wants to remove all the ingredients or share them all. This can be done through the two buttons at the bottom of the page.

Figure 2.10 shows the first design of the shopping list, it displays how much of each ingredient the user needs, it has a simple layout where ingredients are divided into categories so the user has an easy overview of it. Figure 2.11 shows a redesigned version of the shopping list. It is designed to show what recipes each ingredient originates from. This means that if the user needs 2L of milk, they can click on it and see in which recipes they need how much milk. It also makes it easier for the user to remove ingredients if they decide not to make a specific recipe.

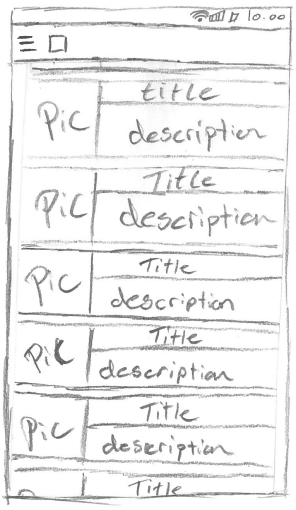


Figure 2.12: The layout for favourites.

As mentioned before, the user can favourite a recipe by clicking the icon when looking at a recipe. Figure 2.12 shows the favourite page, it looks a lot like Figure 2.7. The user removes a favourite recipe either by long clicking the recipe or click it to go into the recipe and click the star symbol to remove the favourite.

2.4 Navigation

There are several different navigation structures that are recommended by Google, the most significant ones being, the navigation drawer and fixed tabs. Both of the design patterns are suitable for switching views frequently, and have multiple top level views. On Google's developer page for Android it says *"Use the navigation drawer if you have more than 3 unique top-level views."*[7], this supports our decision of implementing the navigation drawer in our application. The navigation drawer will also make the application easier to expand with more functionality, and we do not sacrifice vertical space as we would have done with a dedicated tab bar. Vertical space is very important for our application because it is used to display the details of a recipe, search results, and many other things.

Navigation drawer

The navigation drawer is a design that allows the user to navigate to any top level view from any context, which means that even if the user is on a lower level view, the user can just expand the navigation drawer to reach the top level views. The navigation drawer has an option to change which side it should be expanded from, but it is typically expanded from the left side of the application.

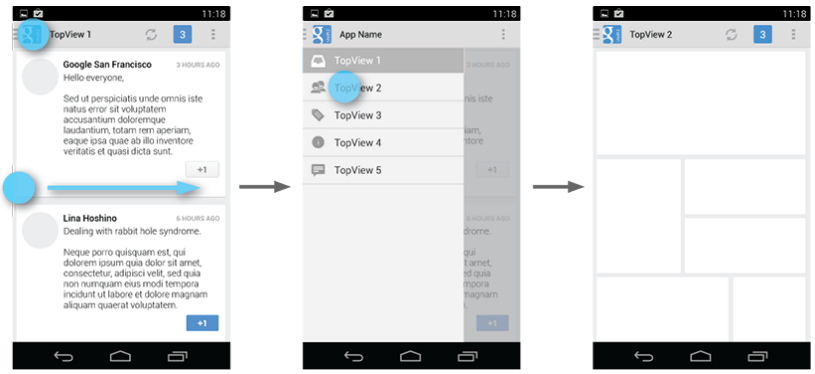


Figure 2.13: Navigation drawer overview[7].

There are two different ways of expanding the navigation drawer, the consistent one being swiping from the left edge of the screen to the centre, this can always be achieved no matter the context. The second one being pressing the button in the top left corner, this option is not consistent since the button changes both appearance and functionality depending on which view level the user is on. If the user is on a top level view, the button's appearance will consist of three thick lines, and Figure 2.13 shows that if pressed the navigation drawer will expand. If the user is entering a lower level view, it will change the button's appearance to an arrow pointing left, and its functionality to be a return button, that if pressed will navigate the user back to the top most view. Each item in the navigation drawer navigates to different top-level pages in the application or typical main actions like sign-in/sign out.

Application navigation

In our application we expect to implement six top level views ingredient search, recipe search, favourites, shopping list, settings, and sign-in/sign out. Our application only has one lower level view which is the display of a recipe, it is a lower level view because it is only accessible through other views. Some views or actions in the application might require the user to be signed in, for them to be available. Like the favourites view, which displays the recipes that the user has favourited. If the user is not signed in when entering the favourite view, they are prompted with a text asking them to sign-in, this should always happen in the case the user tries to use user-functionalities. In the bottom of the navigation drawer we plan to add an item with the action of signing in, or signing out the user, it is located in the bottom of the navigation drawer to signalise its difference with the other items in the navigation drawer.

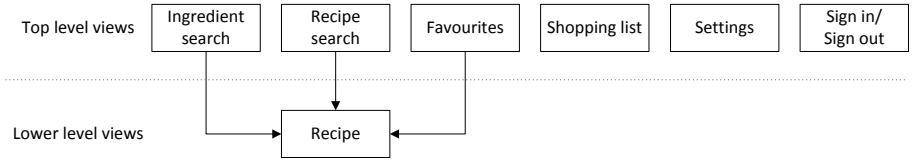


Figure 2.14: Navigation view flow.

Figure 2.14 shows the views that are accessible in the application, and the navigation flow between the views. It can be seen that to reach the recipe view you have to go through either ingredient search, recipe search or favourites. It should be noted that it is always possible reach to top level views, no matter the context the user is in.

2.5 Search by Ingredients

As Requirement 2 and Requirement 3 states, we want two different ways to search for recipes: Free-text and based on ingredients.

The primary feature of the application is the ability to search for recipes based on a number of ingredients. Unlike the existing service, Supercook, described in Section 1.1, we want recipes with the most matching number of ingredients to have a higher precedence than recipes with the least missing ingredients. We believe this will yield a better results since the user is unlikely to enter every single ingredient they own. This should also make the application more useful when searching by items on sale in a supermarket.

When searching by an ingredient we also want recipes where this ingredient is an essential part of the recipe to have a higher precedence than recipes where the ingredient is optional, e.g. when searching for recipes which contain "cream", an ice cream recipe should have a higher precedence than a recipe for making hot chocolate, since cream is not an essential ingredient for making hot chocolate.

The following shows the design of our precedence function we use when searching for recipes:

1. Remove recipes without any matching ingredients.
2. Sort by least missing optional ingredients.
3. Sort by most matching optional ingredients.
4. Sort by least missing mandatory ingredients.
5. Sort by most matching mandatory ingredients.

It should be clarified that latter sorts will be of more importance than earlier sorts, i.e. our result will primarily be sorted by the number of matching mandatory ingredients. Any recipes with an equivalent number of mandatory ingredient will be sorted by the least missing number of ingredients, and so on.

2.6 Database

We have chosen to use a fork of MySQL called MariaDB. The main reason for using a variant of MySQL is because of previous experience with it. MariaDB offers some additional features and some speed improvements compared to MySQL, but the main reason for choosing MariaDB over MySQL is that MariaDB is truly open source whereas MySQL is only released under GPL[12].

Figure 2.15 shows the Entity-Relationship (ER) diagram of our our database. The diagram does not show the attributes of the entities. All functional relationship types are total participants except for the relationship *Parent* between *Categories*.

Table 2.1 shows the mapping of our ER diagram to relations. It also shows the attributes of the entities.

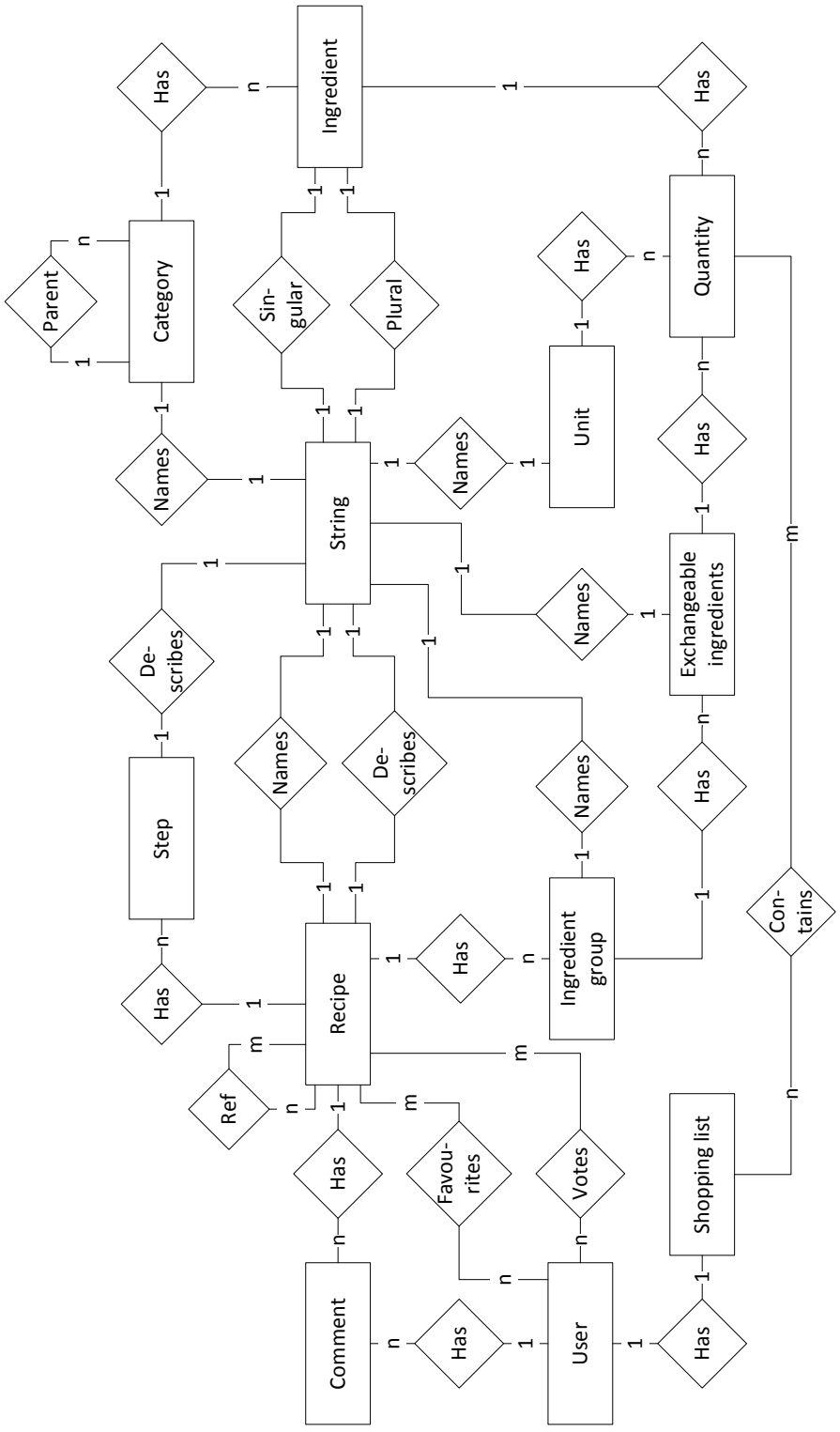


Figure 2.15: Entity-relationship diagram.

string:	[{ id: int, us: text }]
unit:	[{ id: int, metric → string: int, imperial → string: int, conversion: float }]
category:	[{ id: int, name → string: int, parent_id → category: int, image: varchar }]
ingredient:	[{ id: int, singular → string: int, plural → string: int, category_id → category: int }]
recipe:	[{ id: int, name → string: int, description → string: int, image: varchar, upvotes: int, downvotes: int }]
votes:	[{ id: int, recipe_id → recipe: int, user_id → user: int, vote: boolean }]
references:	[{ id: int, ref_a → recipe: int, ref_b → recipe: int }]
exchangeable_ingredients:	[{ id: int, recipe_id → recipe: int, mandatory: boolean, order: int, ingredient_group_id → ingredient_group: int }]
quantity:	[{ id: int, ingredient_id → ingredient: int, exchange_ingredient_id → exchangeable_ingredients: int, amount: double, unit_id → unit: int }]
step:	[{ id: int, description → string: int, image: varchar, recipe_id → recipe: int, order: int }]
comment:	[{ id: int, user_id → user: int, recipe_id → recipe: int, text: text }]
shopping_list:	[{ id: int, user_id → user: id }]
contains:	[{ id: int, shopping_list_id → shopping_list: int, quantity_id → quantity: int }]
user:	[{ id: int, hash: text }]
favourites:	[{ id: int, user_id → user: int, recipe_id → recipe: int }]
ingredient_group	[{ id: int, name → string: int, order: int, recipe_id → recipe: int }]

Table 2.1: Relations.

Server Implementation

3

This chapter describes how the API has been implemented on the server as well as how the search queries are designed and implemented.

3.1 API

The server is implemented as a REST API using GET parameters, this means that whenever a request is made to the server, it is of the following format:

$$http://figz.dk/food/lib/ingredients.php?lang=us$$

In the given example the parameter sent is *lang* with the value of *"us"*.

We have created a single file on the server that handles all database connections, this means that whenever a call is made to the server, the parameters are checked and then sent to an instance of the database class. This way we isolate all connection to the database making it easier to maintain and test. The server will then return a JSON formatted object which can easily be deserialised on the mobile application. We have chosen JSON as a format since it is well supported, easy to generate, and deserialisable in both PHP and Android.

```
1  require_once('DB.php');
2  // Connect to the DB
3  $DB = new DB();
4
5  if (isset($_GET["lang"]))
6      $lang = $_GET["lang"];
7  else
8      $lang = "us";
9
10 $DB->getIngredients($lang);
```

Listing 3.1: ingredients.php.

Lines 1-3 Include the database file and create an instance of the `DB` class. The constructor of the `DB` class will automatically create a connection to the database.

Lines 5-8 Check whether the *lang* parameter is set. If the parameter is not set it will be defaulted to *"us"*.

Line 10 The method `getIngredients()` in the `DB` class is called with the given language parameter.

```
1  public function getIngredients($lang) {
2      if ($this->checkLanguage($lang)) {
3          $ingredientQuery = $this->con->prepare(
4              "SELECT ingredient.id, singular.{$lang} AS singular, plural.{$lang}
                   AS plural
```

```
5                 FROM ingredient
6                 JOIN string AS singular ON ingredient.singular = singular.id
7                 JOIN string AS plural ON ingredient.plural = plural.id
8                 ORDER BY singular.{$lang} ASC");
9             $ingredientQuery->execute();
10            return $this->echoJson($ingredientQuery);
11        }
12    }
```

Listing 3.2: getIngredients() method of DB class.

Line 1 `getIngredients()` takes the language as a parameter.

Line 2 `checkLanguage()` is called to check whether the language is valid. The method simply checks that the string matches a string in an array of valid strings.

Lines 3-8 The SQL query is prepared for execution with the language parameter.

Line 9 The query is executed.

Line 10 The `echoJson()` method is called to print the result of the SQL query.

As seen in Listing 3.2 the method `echoJson()` is called. This method will serialise the result into a JSON formatted object and then print it. An example of result from a request to the server could look like the following:

```
1  [
2      {
3          "id": 7,
4          "singular": "Cointreau",
5          "plural": "Cointreau"
6      },
7      {
8          "id": 1,
9          "singular": "cream",
10         "plural": "cream"
11     },
12     {
13         "id": 2,
14         "singular": "dark chocolate",
15         "plural": "dark chocolate"
16     },
17     ...
18 ]
```

Listing 3.3: Example result from `getIngredients()`.

In Listing 3.3 it is worth noticing that the result is an array of JSON objects where every row that was selected from the database is represented as an object. Since the database contains a lot of ingredients, this example is only a snippet of the total result. The different types of requests the server can take are:

favourites This request takes two required parameters and four optional parameters. The two required parameters are: *action* and *hash*. The four optional parameters are: *recipeid*, *limit*, *offset*, and *lang*.

action This must be set to one of four things: *status*, *add*, *remove*, and *get*. *hash* must be set to the SHA-256 hashed value of the user's email. Depending on the set value of *action* the following will happen:

status Requires *recipeid* to be set. This request will return a boolean value indicating whether the recipe is added to the user's list of favourites or not.

add Requires *recipeid* to be set. This request will add the given recipe to the user's list of favourites .

remove Requires *recipeid* to be set. This request will remove the given recipe from the user's list of favourites.

get Requires *limit*, and *offset* to be set. This request will return all recipes from the user's list of favourites.

ingredientsearch This request takes one required parameter and three optional parameters. The required parameter is *i* and it must be set to an array of ingredient ids. The optional parameters are: *lang*, *limit*, and *offset*. The request will return an array of partial recipes. A partial recipe consists of the recipe ID, name, and image name.

ingredients This request takes one optional parameter: *lang*. If *lang* is not set it will default to "us". The request will return an array of all ingredients in the system in the given language.

recipe This request takes one required parameter and two optional parameters. The required parameter is *id* which must be set to a recipe ID. The optional parameters are *lang* and *metric*. This request will return an object containing all information about the given recipe.

textsearch This request takes one required parameter and three optional parameters. The required parameter is *q* and it must be set to the string of which you want to search for. The optional parameters are *lang*, *limit*, and *offset*. The request will return an array of partial recipes that satisfy the search query.

3.2 Free-text Search

Free-text search can be used to search for parts of the recipe name or words in the recipe description. As defined in Section 2.6, we use MariaDB as our Relational Database Management System (RDBMS). MariaDB has support for full-text search which makes it simple to implement free-text search. MariaDB also has support for a special full-text search called "Boolean search"[11]. In "Boolean search", the user is able to use special operators like +, -, "", and * for marking words as mandatory, discarding rows containing specific word, matching phrases, and inserting wildcards, among other things. Unlike the default search mode, "Natural Language", which returns a score for each row, the "Boolean search" only outputs "true" or "false", to indicate whether a match was found. We want to be able to order the result based on how well the text in the recipes match the query. In particular, a match in a title of a

```
1  SELECT recipe.id, name.{$lang} AS name, image,
2    (MATCH (name.{$lang}) AGAINST ('{$str}')) AS title_score,
3    (MATCH (description.{$lang}) AGAINST ('{$str}')) AS desc_score
4  FROM recipe
5  JOIN string AS name ON name.id = recipe.name
6  JOIN string AS description ON description.id = recipe.description
7  WHERE MATCH (name.{$lang}, description.{$lang})
8    AGAINST ('{$str}* IN BOOLEAN MODE)
9  ORDER BY title_score DESC, desc_score DESC
10 LIMIT {$limit} OFFSET {$offset}
```

Listing 3.4: Free-text search.

recipe should have higher precedence in the result than a match in a description of a recipe. Listing 3.4 shows the SQL query we use for free-text search.

It should be noted that variables are only inserted directly in the SQL queries for clarification. Inserting user specified variables directly in an SQL query will make the query vulnerable to SQL injections. In the actual implementation, the SQL queries are protected using PHP's *prepared statements*. In prepared statements, the query and variables are separated to make sure that the values of the variables are not executed on the RDBMS. Prepared statements can also be used to execute a query repeatedly with different values, without sending the entire query to the RDBMS every time.

Since SQL is a declarative language, each individual SQL query's clauses will not be explained sequentially but in an order that is easier to understand.

Lines 4-6 The `recipe` table is joined with the `string` table two times to get both the title and description of each recipe.

Lines 7-8 Keep only recipes which have a match in `BOOLEAN MODE`. Note the asterisk, '*', which is appended to the user defined search text. This allow the user to enter e.g. "choco" and get a match on e.g. "Chocolate cake". We would like to add a prefixed asterisk as well to allow users to search for e.g. "berry" and get results like "Strawberry Pie". Unfortunately it is not possible due to limitations in MariaDB[11].

Lines 2-3 When you use full text search in the default mode, `NATURAL LANGUAGE MODE`, it calculates a score which reflect the relevance of the match. A high relevance could indicate that the matched query represent a big part of the text. The scores for when the user defined text is matched to the recipes' titles and descriptions are calculated using `NATURAL LANGUAGE MODE`. These values are stored in respectively `title_score` and `desc_score`.

Line 9 The result is ordered by the calculated scores with `title_score` having a higher precedence than `desc_score`.

Line 10 The result is limited and offset according to the application's needs i.e. how far the user has scrolled through the result.

Line 1 The ID, name, image, and scores of the recipes are returned from the query and then later sent to the application.

3.3 Search by Ingredients

The application also allows the user to search for recipes by entering a number of ingredients. The search algorithm should then find recipes which contain one or more of the ingredients specified by the user. The search algorithm should also order the results according to the precedence function described in Section 2.5.

The source code for the search algorithm, excluding SQL queries, is available in Appendix B. The general structure of the algorithm is as follows:

1. The input parameters are validated.
2. A search query is sent to the RDBMS together with the input parameters. The result is an ordered list of recipes.
3. For each recipe we find the exchangeables that are not *covered* by the user specified ingredients. Exchangeables allow us to specify which ingredients in a recipe can be replaced by other ingredients. An example could be a recipe where either minced beef or minced pork could be used. If the user has specified either one in the search query we consider the exchangeable to be *covered* i.e. the user does not need to buy minced pork if he searched for "minced beef".
4. Using the list of exchangeables that are not covered, we for each recipe create a list containing the ingredients they also need in order to follow the recipe. This list excludes the ingredients the user specified for the search.
5. The result is returned to the user.

The search query mentioned in step 2 clearly does most of the work in the search algorithm by returning an ordered list of recipes related to the ingredients specified by the user. The query itself has the following structure:

1. Calculate how many of the user specified ingredients exists in each recipe.
2. Calculate how many not covered exchangeables exists in each recipe.
3. Retrieve the name, description, and image for each recipe and order them according to the precedence function described in Section 2.5.

Each part of the search query will be described in detail on the following pages.

$quantityQuery

Listing 3.5 shows the SQL query `$quantityQuery`, that returns every quantity that contains one of the ingredients specified by the user.

Lines 2-3 The quantities that consist of one of the ingredients defined by the user are retrieved. `FIND_IN_SET` allow us to check a comma separated string instead of a tuple which is required by the `IN` operator. It would not be possible to provide a tuple to the `IN` operator because PHP's prepared statements does not have support for a variable number of arguments.

```
1 SELECT DISTINCT exchange_ingredient_id, ingredient_id
2 FROM quantity
3 WHERE FIND_IN_SET(ingredient_id, '{$commaSeparatedIngredients}')
```

Listing 3.5: $quantityQuery, return quantities that match the search.

Line 1 The IDs of the found pairs of exchangeables and ingredients are returned.

$ingredientQuery

Listing 3.6 shows the SQL query `$ingredientQuery`, that returns the number of times the individual ingredients match in each recipe.

```
1 SELECT recipe_id, ingredient_id,
2   SUM(mandatory) AS mandatory_matching,
3   SUM(NOT mandatory) AS optional_matching
4 FROM ( {$quantityQuery} ) AS temp
5 JOIN exchangeable_ingredients
6   ON temp.exchange_ingredient_id = exchangeable_ingredients.id
7 GROUP BY recipe_id, ingredient_id
```

Listing 3.6: $ingredientQuery, returns the number of times the individual ingredients appear in each recipe.

Lines 4-6 The result of the `$quantityQuery` is joined with the `exchangeable_ingredients` table.

Line 7 The result of the join is then grouped by `recipe_id` and `ingredient_id`.

Lines 1-3 The IDs of the pairs of recipes and ingredients are returned together with the sum of the `mandatory` column in `exchangeable_ingredients`. Since the `mandatory` column consists of boolean values we will simply get the number of times this ingredient appear in the recipe as mandatory. Likewise, we get the number of times this ingredient appear in the recipe as optional.

$matchingQuery

```
1 SELECT recipe_id,
2   COUNT(NULLIF(mandatory_matching,0)) AS mandatory_matching,
3   COUNT(NULLIF(optional_matching,0)) AS optional_matching
4 FROM ( {$ingredientQuery} ) AS temp
5 GROUP BY recipe_id
6 LIMIT {$limit} OFFSET {$offset}
```

Listing 3.7: $matchingQuery, returns the matching ingredients count for each recipe.

Listing 3.7 shows the SQL query $matchingQuery, that returns the number of matching ingredients for each recipe.

Lines 4-5 The result of $ingredientQuery is grouped by the recipe_id.

Line 6 The result is limited and offset according to the applications needs i.e. how far the user has scrolled through the result.

Lines 1-3 The ID of each recipe, and the number of matching mandatory and optional ingredients are returned. To calculate this, we first replace the matching count with NULL if the count is 0. Since NULL values are ignored by the COUNT function, we will get the number of unique ingredient matches.

$lackingQuery

Listing 3.8 shows the SQL query $lackingQuery, that returns the number of lacking ingredients for each recipe. Retrieve all exchangeables that are not covered by an ingredient specified by the user or an ignored ingredient.

```
1  SELECT recipe_id,
2    SUM(mandatory) AS mandatory_lacking,
3    SUM(NOT mandatory) AS optional_lacking
4  FROM exchangeable_ingredients
5  WHERE NOT EXISTS (
6      SELECT *
7      FROM quantity
8      WHERE quantity.exchange_ingredient_id = exchangeable_ingredients.id
9      AND (
10         FIND_IN_SET(ingredient_id, '{$commaSeparatedIngredients}')
11         OR ingredient_id IN ( {$ignoredIngredients} )
12     )
13 )
14 GROUP BY recipe_id
```

Listing 3.8: $lackingQuery, returns the number of lacking ingredients for each recipe.

Lines 4-13 Retrieve all exchangeables that are not covered by an ingredient specified by the user or an ignored ingredient.

Lines 7-8 Retrieve quantities which are a part of a given exchangeable.

Lines 9-12 Discard the exchangeable if any of its consisting quantities contain one of the ingredients specified by the user or an ignored ingredient.

Line 14 The resulting exchangeables are grouped by the recipes they are a part of.

Lines 1-3 The ID of each recipe, and the number of lacking mandatory and optional exchangeables are returned.

$sortingQuery

Listing 3.9 shows the SQL query $sortingQuery, that returns a list of recipes sorted according to the precedence function defined in Section 2.5.

```
1 SELECT recipe.id, name.{$lang} AS name, image
2 FROM ( {$matchingQuery} ) AS temp1
3 LEFT JOIN ( {$lackingQuery} ) AS temp2 ON temp1.recipe_id = temp2.
      recipe_id
4 JOIN recipe ON recipe.id = temp1.recipe_id
5 JOIN string AS name ON name.id = recipe.name
6 ORDER BY mandatory_matching DESC, mandatory_lacking ASC,
7   optional_matching DESC, optional_lacking ASC
```

Listing 3.9: $sortingQuery, combine and sort.

Lines 2-3 The result of $matchingQuery is joined with the result of $lackingQuery. A LEFT JOIN is used since $lackingQuery will not return rows of recipes that are completely covered by the ingredients specified by the user and ignored ingredients.

Lines 4-5 The rows are then joined with the recipe table and the string table.

Lines 6-7 The result is ordered as defined in Section 2.5. In descending order of precedence: most mandatory ingredients matching, least mandatory exchangeables lacking, most optional ingredients matching, least optional exchangeables lacking. It should be noted that the LEFT JOIN in Line 3 will insert NULL values in any recipe that is fully covered. In our case, the NULL values represent zeros since they occur when a recipe have zero exchangeables that are not covered. We do not need any extra clauses to handle those NULL values since NULL values always have the highest precedence when sorted.

Line 1 The ID, name, and image of the recipes are returned from the query and then later send to the application.

Get needed ingredients

We now have all the recipes returned by the search query but we also want to provide the user with a list of ingredients they also need in order to follow the recipes, i.e. the ingredients in the exchangeables that are not covered by the ingredients specified by the user. To find those ingredients we find the exchangeables that are not covered using the query defined in Listing 3.10.

Line 2-3 The exchangeables which are a part of a specified recipe are retrieved.

Line 4 Optional exchangeables are discarded.

Lines 5-13 Discard exchangeables which are covered by the ingredients specified by the user.

```
1  SELECT id
2  FROM exchangeable_ingredients
3  WHERE exchangeable_ingredients.recipe_id = {$recipeId}
4  AND mandatory = 1
5  AND NOT EXISTS (
6      SELECT *
7      FROM quantity
8      WHERE exchangeable_ingredients.id = quantity.exchange_ingredient_id
9      AND (
10         FIND_IN_SET(ingredient_id, {$ingredients})
11         OR FIND_IN_SET(ingredient_id, '{$ignoredIngredients}')
12     )
13 )
```

Listing 3.10: Query to find the exchangeables that are not covered.

Lines 7-8 Retrieve quantities which are a part of a given exchangeable.

Line 9-12 Discard quantities which does not contain one of the ingredients specified by the user or an ignored ingredient.

Line 6 If the query returns any rows it means that the exchangeable is covered by any of the ingredients specified by the user.

Line 1 The IDs of the resulting exchangeables are returned. These exchangeables are not covered by any user specified ingredients.

We now have the exchangeables for each recipe that are not covered by an ingredient specified by the user or an ignored ingredient. We then simply find the ingredients that are a part of those exchangeables using the query in Listing 3.11.

```
1 SELECT DISTINCT ingredient_id
2 FROM quantity
3 WHERE exchange_ingredient_id = ?
```

Listing 3.11: Query to get the ingredient of an exchangeable, excluding ignored ingredients.

Lines 2-3 The quantities which are a part of a specified exchangeable are retrieved.

Line 1 The ingredient IDs of the resulting quantities are returned.

Application Implementation

4

This section explains and shows the final design of the application. The final design is a result of all the design analysis done in Chapter 2. It should be noted that compared to the prototypes shown in Section 2.3 the shopping list has not been implemented.

4.1 Final Application Design

The first time the user opens the application they are prompted with a navigation drawer, as seen on Figure 4.1. The reason this is shown the first time the application is opened is to inform the user that it is available, as this is the tool they will be using to navigate between the different pages in the application. The navigation drawer is available from every page of the application.

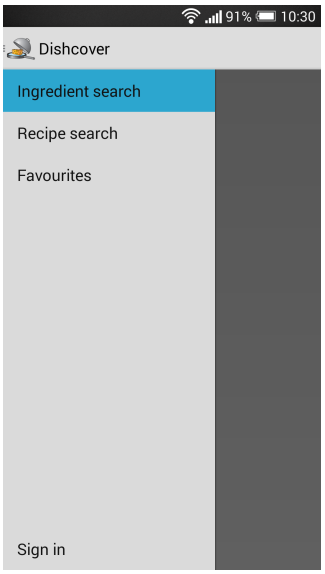


Figure 4.1: Navigation drawer.

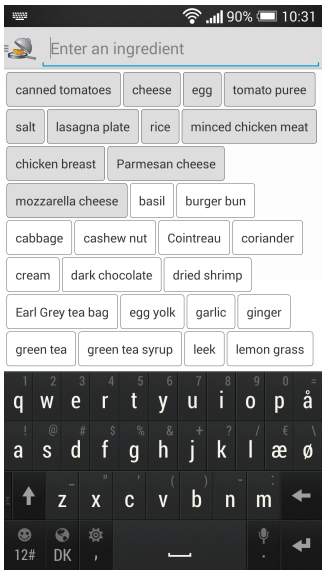


Figure 4.2: Word cloud.

The user opens the navigation drawer either by swiping from the left edge of the screen or by clicking the icon in the top left corner and it is closed either by clicking outside of the navigation drawer, by clicking the icon in the top left corner again, by swiping from right to left anywhere on the screen, or by pressing back.

If the user closes the navigation drawer and presses the search field on the action bar where it says “Enter an ingredient”, the word cloud pops up. The word cloud is preloaded with all the ingredients in alphabetic order and they can easily be clicked and added to the search. The word cloud is static even when the user add ingredients to the search. The user is also able to search for

ingredients by typing in the name. When the user types, a list of suggestions will come up and the user can either press a suggestion to add the ingredient to the search, or press enter to add the first ingredient to the search. The details of the search suggestions are explained later.

Figure 4.2 shows the word cloud with a number of ingredients added. The grey labels are included in the search, whereas the white labels are only suggestions the user can press and are not included in the search. The user is not yet able to filter ingredients by specifying allergies. The user is therefore currently able to find recipes that uses ingredients that the user is allergic to.

Figure 4.3: List of recipes.

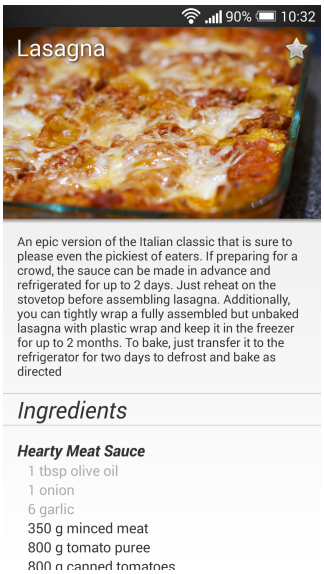


Figure 4.4: Recipe layout.

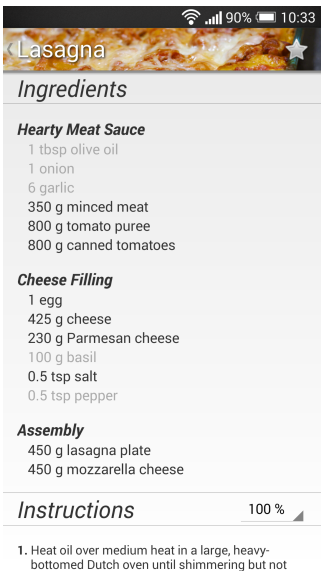


Figure 4.5: Recipe layout.

Figure 4.6: Recipe layout.

If the user presses the back button or presses the enter key without having typed in anything into the search field, a search is performed. Figure 4.3 shows the list of recipes based on the ingredients entered in Figure 4.2. The search field changes its text after a search, it shows some of the ingredients that the user has included in the search, the search field itself works like before. If the ingredient search exceeds the number of recipes that can be shown on the screen, they will automatically be downloaded as the user scrolls down. This also applies to the list of recipes in recipe search and in favourites.

The yellow text on some of the recipes indicates what ingredients the user is missing in order to have all mandatory ingredients for the recipe.

If the user presses a recipe it will open and they will see the full recipe. This can be seen on Figure 4.4, Figure 4.5, and Figure 4.6. The three figures shows an example of a full recipe. The grey ingredients are optional, where as the black ingredients are mandatory.

Figure 4.5 shows the list of ingredients needed to make the recipe, listed in groups. Figure 4.6 shows the instructions in order to make the recipe, the user is able to change the font size in the top right corner. The user is also able to favourite a recipe by pressing the star in the top right corner. In order to favourite a recipe the user needs to be signed in.

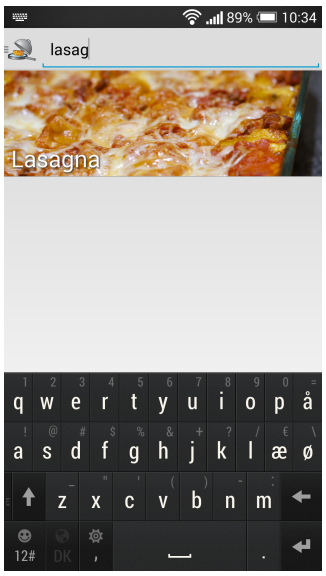


Figure 4.7: Recipe search.

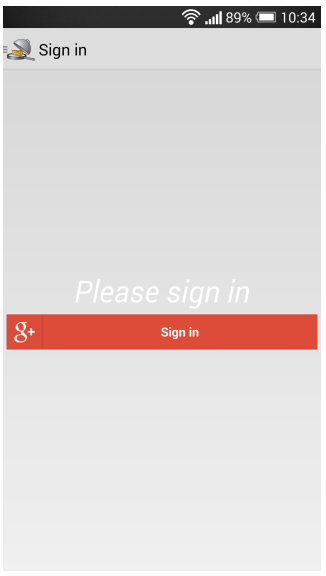


Figure 4.8: Sign-in.

The user is not just limited to searching for recipes based on ingredients, they can also search for recipes using free text, this is shown in Figure 4.7. This allows the user to search for a recipe based on its name or words included in the recipe description.

When a user wants to access their favourites, they have to be signed in. In case the user tries to access their favourite recipes without being signed in, they are prompted with a page asking them to log in using Google+, this is shown in Figure 4.8.

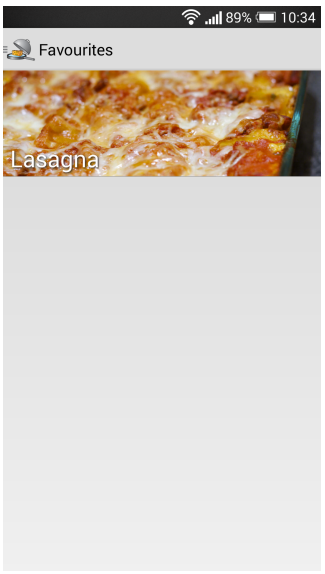


Figure 4.9: Favourite list.

After the user has logged in, they are able to access their favourite recipes as shown in Figure 4.9. The user is able to click the recipes and open the full recipe. In order to remove recipes from favourites, the user can either long click or open the recipe and press the star again.

4.2 Search Suggestion

We want the users to be able to enter the ingredients they have in their home, however it can be quite tedious to type in the full ingredient names of all the ingredients, so we want to aid the users by providing them with search suggestions as they type in the names. The user can press these search suggestions to quickly add the ingredient to the list of ingredients that the user has. We have decided that only the ingredients that are used in recipes can be added to list of ingredients that the user has. This means that the user is only able to enter ingredients that is already used by a recipe. The reason for this was that we did not want the user to be able to search for random text strings, or to get an empty search because none of their ingredients matched those in the recipes.

When the users opens the application, all the ingredients are loaded into a list, this list of ingredients is then used for the search suggestions. Each time the user types in a letter, the input string is run through a regular expression trying to match the input string to an ingredient name in the list, the matching ingredients is added to a list and shown on the screen as a search suggestion. This can be seen in Listing 4.1.

```
1 Pattern p = Pattern.compile("(^|\\s)" + query);
2
3 for (Ingredient ingredient : allIngredients) {
4     Matcher matcher = p.matcher(ingredient.getSingular().toLowerCase());
5
6     if (matcher.find()) {
7         suggestionName.add(ingredient.getSingular());
8     }
9 }
```

Listing 4.1: Search suggestions.

Line 1 `Pattern.compile()` returns a compiled form of the regular expression. The expression matches the start of a string or a space and then the input itself.

Line 3-9 We loop through the list of ingredients to see if the regular expression matches any of the ingredient names.

Line 6-7 Each, if any, suggestion that is found is added to the suggestion list.

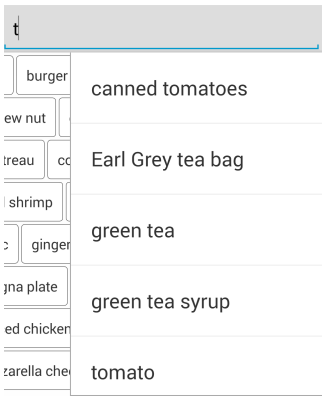


Figure 4.10: Suggestions with a "t".

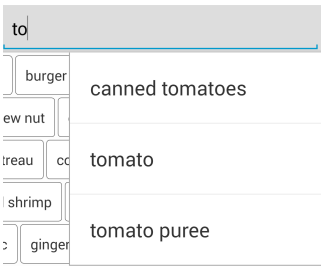


Figure 4.11: Suggestions with "to".

Figure 4.10 and Figure 4.11 shows two examples of how this works. When the user types in a "t" they get all ingredients where a part of the string starts with a "t", if the users continues and types an "o", the suggestions are updated and now only show the words where parts of the string starts with "to".

The user can add ingredients to the word cloud in two ways, they can either type parts of the name as shown in Figure 4.10 and Figure 4.11 and press the correct suggestion in the list of suggestions or they can type parts, or the full name, and press the enter button on the keyboard. Doing this will add the first ingredient in the suggestion list to the search.

4.3 Authentication

We have chosen to implement a social media sign-in option because it does not require us to handle the user's passwords and other personal information. With a social media sign-in implemented in our application as a way of identifying the user, we move the security measures to the specific social media. There are many different social media sign-in options such as Google+, Facebook, Twitter, etc. We chose Google+ to be the first social media sign-in to implement. Other sign-in options could later be implemented in the application. It should be noted that it is not necessary for the user to be signed in to search for recipes, but actions like favouriting, adding to items to the shopping list, requires them to be signed in.

It makes sense to use Google+ sign-in since Google recommends Android users to have a Google account, because otherwise the user would limit their experience with an Android device. The Google+ sign-in authenticates the user and manages the OAuth 2.0 flow, which simplifies the integration with the Google Application Programming Interface (API). The Google+ sign-in process can seamlessly authenticate a user without them having to enter a password or even an account name. Once granted access, the application is able to access to user's Google+ information through their Google+ account[17].

In the implementation of the Google+ sign-in we use an object called `GoogleApiClient`[19], which is provided by the Google Play services application. The object `GoogleApiClient` should be initialised and connected separately in every activity, so each activity has its own life cycle with the `GoogleApiClient`, Figure 4.12 shows the basic life cycle of a `GoogleApiClient` object in an activity.

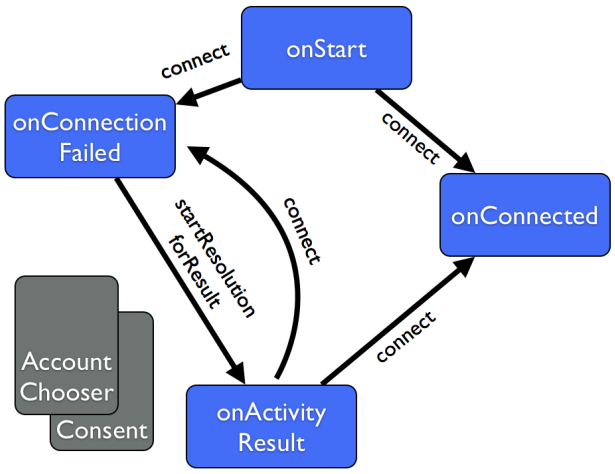


Figure 4.12: The `GoogleApiClient` basic life cycle[18].

We have implemented an activity called LogInActivity, which implements the sign-in features. Every activity in the application extends this activity, which means that all activities in the application has the sign-in feature. The blue boxes represents methods in the LogInActivity, and the two grey boxes

represents other activities. The method `onStart()` is invoked at activity start up, and we immediately try to sign-in. This can either succeed or fail, if it fails `onConnectionFailed()` will be called, this will happen if the user has not yet consented with the Google+ sign-in, the application now awaits for them to press the sign-in button. When the user presses the sign-in button and they have multiple Google accounts, the application opens an account chooser activity. After selecting the account that the user want to use, a consent activity will show presenting them with the Google+ information that the application should be allowed to access. When the user has granted the application access, then the result of the consent activity is returned to `onActivityResult()`, and the user is signed in. If the user did not grant access to the application the routine starts all over. When the user is connected `onConnected()` is called, this method simply updates the UI and downloads information from the user's Google+.

When the user is signed in to our application we can gain information of the user by using the `GoogleApiClient`. In our application we only use the user's Google email(account name) and their real name(for comments on recipes). When we receive the user's email we hash it using the SHA-256 hashing algorithm. We hash it because we have no use of the email other than its uniqueness, and from a security aspect it does not make sense to store their email in plain text both in the application and database. Every time the user makes an action that requires them to be signed in, we send their hashed email to the server together with the action.

4.4 Model Component

The application needs a model to represent recipes. The recipe data we receive from the server is in JSON format, the data is then parsed to our application model. The model component is shown in Figure 4.13.

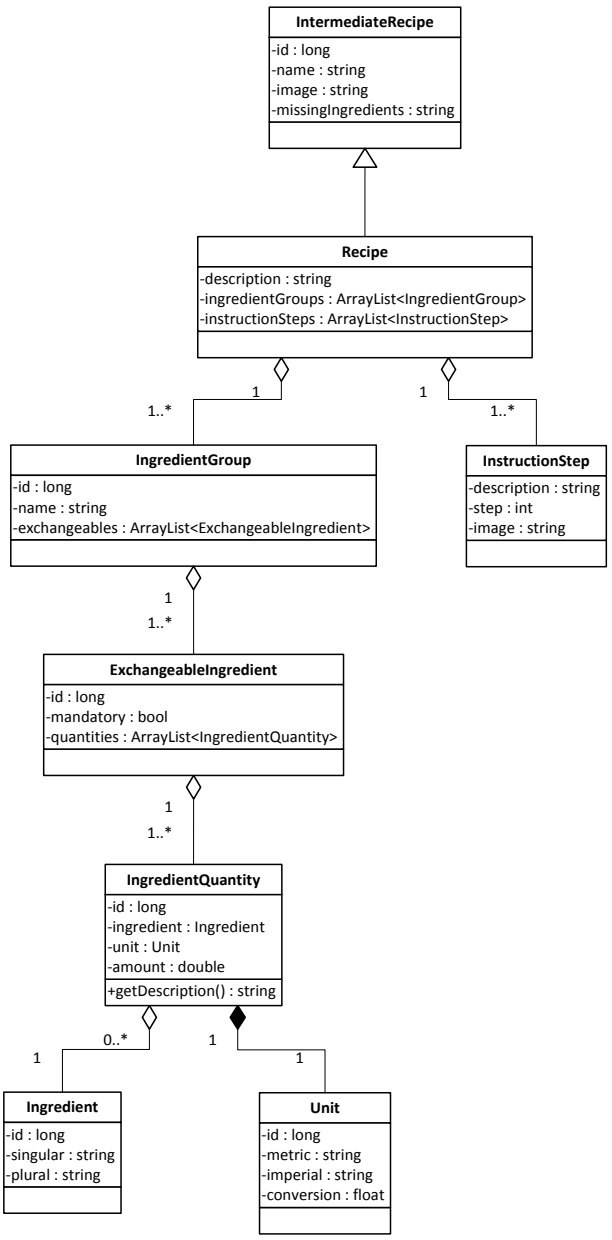


Figure 4.13: Application model.

We have two different representation of a recipe, `Recipe` and `IntermediateRecipe`.

The IntermediateRecipe is a small subset of a recipe which is used to display search results. When the user clicks one of the search results, then the rest of a Recipe is downloaded from the server and displayed to the user.

A Recipe consists of one or more instruction steps and one or more IngredientGroups which is a grouping of ingredients. An ExchangeableIngredient consists of one or more IngredientQuantity class which are interchangeable.

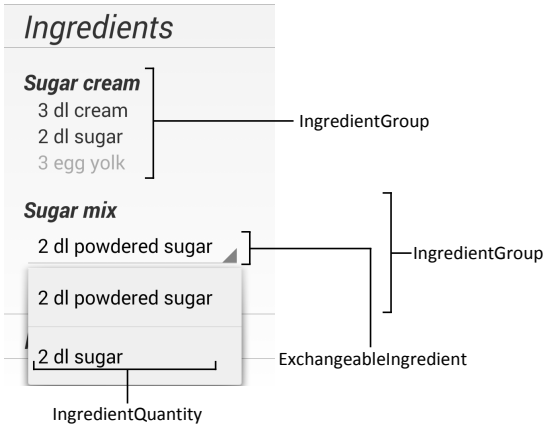


Figure 4.14: Ingredient model.

The use of ingredients are easier understood by looking at Figure 4.14. If an ingredient can be exchanged by another ingredient it can be clicked, which will reveal the different ingredients that can be used. It can be seen that *2 dl powdered sugar* is exchangeable with *2 dl sugar*.

The reason why the attributes from Unit are not modelled inside IngredientQuantity is because the model resembles our relational database as much as possible. The Ingredient class could also have been modelled inside IngredientQuantity, but Ingredient is used when searching by ingredients.

4.5 Server Communication

In order to communicate with the server we created an abstract class which must be inherited for each different request we make to the server. The class inherits from AsyncTask which makes it easy to create a new task on a new thread and send the result to the User Interface (UI) thread.

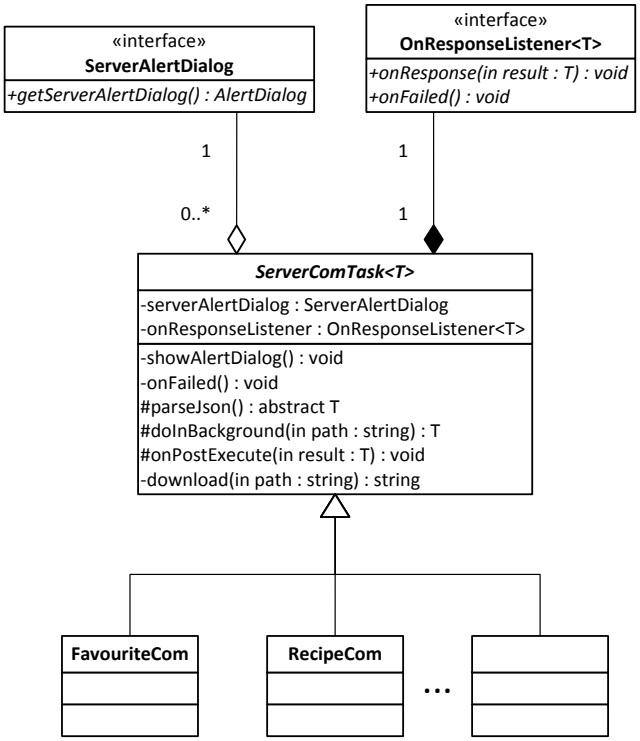


Figure 4.15: ServerComTask.

ServerComTask is a generic class that declares a type which the abstract method ParseJson() must return. The object retuned by ParseJson() is created by parsing the JSON it is supplied with.

ServerAlertDialog is a simple interface which returns an AlertDialog, which the ServerComTask can use to display an alert, if something went wrong. The interface is implemented in all activities so all server requests can share the AlertDialog reference. We use it to display a static message that notifies the user that the connection with the server failed.

OnResponseListener is a generic interface. It is used by ServerComTask to communicate the result to where the server request was started. It sends the generic type as the result with onResponse(). If anything went wrong with the request and no answer was produced, then onFailed() is called so the lack of a result can be handled.

We have several different request we can make from the application to the server:

FavouriteCom Gets or sets whether a recipe is favourited by a specific user.

FavouriteListCom Gets the list of favourites for a specified user.

IngredientCom Gets a list of all ingredients so the user can use them to search for recipes by ingredients.

IngredientSearchCom Searches for recipes based a user specified ingredients.

RecipeCom Gets an entire recipe.

RecipeSearchCom Searches for recipes based on a user specified text string.

```
1  public abstract class ServerComTask<T> extends AsyncTask<String, Integer,
       T> {
2      private ServerAlertDialog serverAlertDialog;
3      private OnResponseListener<T> onResponseListener;
4
5      protected abstract T parseJson(String json) throws Exception;
6
7      public ServerComTask(String apiPath, ServerAlertDialog serverAlertDialog,
           OnResponseListener<T> onResponseListener) {
8
9          this.apiPath = apiPath;
10         this.serverAlertDialog = serverAlertDialog;
11         this.onResponseListener = onResponseListener;
12
13         this.execute(apiPath);
14     }
15
16     @Override
17     protected T doInBackground(String path) {
18         try {
19             String response = this.download(path);
20             return this.parseJson(response);
21         } catch (Exception e) {
22             return null;
23         }
24     }
25
26     @Override
27     protected void onPostExecute(T result) {
28         if (result == null) {
29             this.onFailed();
30             return;
31         }
32         this.onResponseListener.onResponse(result);
33     }
34
35     private void onFailed() {
36         this.onResponseListener.onFailed();
37         this.showAlertDialog();
38     }
39 }
```

Listing 4.2: Simplified code of the ServerComTask.

Lines 7-14 The constructor for ServerComTask takes a ServerAlertDialog to display errors, and an OnResponseListener to send the result to.

Line 13 The call to `execute()` starts the overridden abstract method `doInBackground()` on a new thread. We chose to execute the task when the class is instantiated, since the task can only be executed once per instance of the task, so it is required that the path to download from is specified in the constructor.

Lines 16-24 `doinBackground()` must return the generic type. The result is then passed to the method `onPostExecute()` which runs on the UI thread.

Line 19 First we download the data from the server. The download is an *http get* request, so any parameters to the call is specified in the path. The `download()` method is not shown.

Line 20 The response JSON is parsed to the generic type and returned to the UI thread.

Lines 28-31 If the result of the task is `null` then the task failed. `onFailed()` handles this.

Line 32 The generic type result is sent to the response listener, and the execution of the task is complete.

Lines 35-38 If the task failed, then we notify the `OnResponseListener` that the task failed and we show an alert to the user.

The different server request implementations are not shown, they only specify a generic type and then parse the supplied JSON to the generic type. An example of how to make a request on the server and then handle the response is shown in Listing 4.3.

```
1  new RecipeSearchCom((ServerAlertDialog) this.getActivity(), query, new
       OnResponseListener<ArrayList<IntermediateRecipe>>() {
2
3      @Override
4      public void onResponse(ArrayList<IntermediateRecipe> result) {
5          RecipeSearchFragment.this.displayRecipeList(result);
6          RecipeSearchFragment.this.progressCircle.setVisibility(View.GONE);
7      }
8
9      @Override
10     public void onFailed() {
11         RecipeSearchFragment.this.progressCircle.setVisibility(View.GONE);
12     }
13 });
```

Listing 4.3: Search for recipes by text.

Line 1 We make a new instance of `RecipeSearchCom` and create an inline implementation of the `OnResponseListener`. Both the class and the interface uses the generic type `ArrayList<IntermediateRecipe>` which is the result of the request. A `query` string is specified which is the recipe text to search for.

Lines 5-6 When the request is done we display the recipe search results in `RecipeSearchFragment.this`, and hide the progress circle.

Line 11 If the request failed then we must make sure to hide the progress circle as well, the `ServerComTask` will show an alert to the user.

Test

5

This chapter shows the tests performed on our system and the results. We have performed black-box and unit testing in order to validate our system. First part of the chapter describes our black-box tests, showing the test cases and the results. The last part shows the unit testing and the results.

5.1 Black-box

We have created some black-box test cases for our Android application. We are performing the tests ourselves. We do this in order to test that all the different functionality requirements are working and producing correct results.

All test cases are presented in the form of a case header consisting of the case name, objective of the test, preconditions for the test, and the requirement we are satisfying. After the header we have made a procedure describing the steps that should be performed in the case and for each step we have a success criteria describing the expected outcome of the step.

The preconditions specifies whether you should be signed in or not. If not specified, then it does not matter if you are signed in or not. This indirectly fulfils Requirement 11 saying that login should not be required.

Navigation drawer

Case name	Navigation drawer
Objective	Navigate the application using the navigation drawer
Preconditions	
Requirement	

Step	Procedure	Success Criteria
1	Open the Navigation drawer	The Navigation drawer opens
2	Tab the first item in the list	The item's page opens
3	Repeat step 1-2 for all items in the list	The different item pages opens

Recipe search

Case name	Recipe search
Objective	Search and find recipe
Preconditions	Connected to the Internet, application opened
Requirement	3 Recipe search

Step	Procedure	Success Criteria
1	Open the navigation drawer and tab the Recipe search item	The recipe search page opens
2	Press the search field	A keyboard should appear
3	Write parts of a recipe name	Recipes matching the text should appear
4	Write the full name of a recipe	Only recipes matching the full name should appear
5	Write random letters that does not match anything	Nothing should show up

Ingredient search

Case name	Ingredient search
Objective	Search and find recipe
Preconditions	Connected to the Internet, application opened
Requirement	2 Ingredient search

Step	Procedure	Success Criteria
1	Press the search field	A keyboard should appear
2	Type in different ingredients	The ingredients should be added to the search and the word cloud changes to relevant suggestions each time an ingredient is added
3	Search for recipes by pressing the back arrow	The keyboard and word cloud disappears and a list of relevant recipes is shown

Shopping list

Case name	Shopping list
Objective	Add/remove items to/from the shopping list from the shopping list page
Preconditions	User signed in
Requirement	6 Shopping list

Step	Procedure	Success Criteria
1	Open the navigation drawer and tab the Shopping list item	The Shopping list page opens
2	Press the search field	A keyboard should appear
3	Write an ingredient and press enter	The ingredient should appear in the shopping list
4	Select an ingredient	Ingredient is selected
5	Press remove selected items	The selected items should be removed from the list

Shopping list, recipe page

Case name	Shopping list, recipe page
Objective	Add/remove items to/from the shopping list from the recipe page
Preconditions	Application is open, and user signed in
Requirement	

Step	Procedure	Success Criteria
1	Search for and choose a recipe	Recipe page opens
2	Tab the Shopping cart to add an ingredient to the shopping list	A message appears saying the ingredient was added to the shopping list
3	Open the navigation drawer and tab the Shopping list item	The Shopping list page should appear with the added ingredient
4	Navigate back to the recipe	The same recipe should appear and the same ingredients should be added to the Shopping cart
5	Press the shopping cart to remove the ingredient from the shopping list	A message appears saying the ingredient was removed from the shopping list
6	Open the navigation drawer and tab the Shopping list item	The Shopping list opens and the ingredient is removed

Shopping list sharing

Case name	Shopping list sharing
Objective	Share a shopping list
Preconditions	Ingredients has been added to the shopping list, and user signed in
Requirement	7 Sharing

Step	Procedure	Success Criteria
1	Open navigation drawer and tab the Shopping list item	Shopping list page opens
2	Select the ingredients you want to share	Ingredients are selected
3	Press the “Share” button	A menu comes up asking which method you want to use
4	Choose share by SMS	The recipient receives a text message with the shared ingredients
5	Repeat step 2 through 4 Facebook, Google+ and E-mail	The recipient receives a message with the shared ingredients

Recipe sharing

Case name	Recipe sharing
Objective	Share a recipe
Preconditions	A recipe has been selected, and user signed in
Requirement	7 Sharing

Step	Procedure	Success Criteria
1	Press the "Share" button	A list comes up asking what method you want to use
2	Choose share by SMS	The recipient receives a text message with a link to the shared recipe
3	The recipient clicks the link	The recipe opens in a web browser
4	Repeat step 2 through 4	The recipient receives a link to the shared recipe.

Search filter

Case name	Search filter
Objective	Filter the recipe search
Preconditions	Connected to the Internet, added an ingredient to the search filter
Requirement	4 Search filter

Step	Procedure	Success Criteria
1	Search for a recipe using text	A list of recipes is shown
2	Check each recipe	None of the recipes may contain the filtered ingredient
3	Open navigation drawer and tab the "Ingredient search" item	The ingredient search page opens
4	Add different ingredients to the search	A list of recipes shows up
5	Check each recipe	None of the recipes may contain the filtered ingredient
6	Remove all the ingredient and only search for the filtered ingredient	No recipes shows up

Favourites

Case name	Favourites
Objective	Add/remove favourite recipes
Preconditions	Connected to the Internet, and user signed in
Requirement	5 Favourites recipes

Step	Procedure	Success Criteria
1	Open navigation drawer and tap the Search item	The Search page opens
2	Search for, and choose a recipe	The recipe page opens
3	Tap the star icon in the action bar	A message appears saying the recipe has been favourited
4	Open navigation drawer and tap the Favourite item	The favourite page opens. The page contains the recipe that was favourited
5	Tap the recipe which was just favourited	The recipe page opens
6	Click the star icon in the action bar	A message appears saying the recipe has been removed from favourited
7	Open navigation drawer and tap the Favourite item	The favourite page opens. The page does not contain the recipe that was removed from favourites
8	Repeat steps 1 through 4, and long click the favourite recipe	A message appears asking if you would like to remove the favourite
9	Confirm the favourite removal	The favourite recipe is removed

No Internet

Case name	No Internet
Objective	Use application without Internet
Preconditions	Not connected to the Internet, and user signed in
Requirement	

Step	Procedure	Success Criteria
1	Open the application	You get the message "Server connection failed"
2	Search for recipes, trying both ingredient search and recipe search	You get the message "Server connection failed"
3	Connect to the Internet, search and find a recipe, disconnect from the Internet, and try to open a recipe	You get the message "Server connection failed"
4	Connect to the Internet, open a recipe, disconnect from the Internet, and press the favourite button	You get the message "Server connection failed"
5	Open the favourites page	You get the message "Server connection failed"
6	Open the shopping list	The shopping list is displayed

Case results

Navigation drawer This test was a success. All pages opens when clicked in the navigation drawer. Except for the *Sign in* button, it signs you in instead of opening the sign in page.

Recipe search This test was a success. Searching for *cream* returned two recipes. One of the recipes did not contain cream in the name of the recipe, this is because it also searches for recipes with the word *cream* in its description. Searching for a recipe's full name only returned that recipe, and searching for random letters did not return anything.

Ingredient search This test was a success. When writing ingredient names, suggestions appear that match the text entered. The word cloud did not suggest new ingredients based on the ingredients already entered, this features is missing. The recipes also returned contained the ingredients entered.

Shopping list Feature is not implemented.

Shopping list, recipe page Feature is not implemented.

Shopping list, sharing Feature is not implemented.

Recipe sharing Feature is not implemented.

Search filter Feature is not implemented.

Favourites This test was a success. As long as you are signed in this test is a success. The application will ask you to sign in if you are not. Instead of showing a message when you favourite a recipe or remove a favourite recipe, the star changes its appearance indicating favourite is on or off. When a recipe in the favourite list is long clicked the user is able to remove it.

No Internet This test was a success besides the shopping list which is not implemented.

5.2 Unit Testing

The mobile application is designed to be a thin client that sends and receives messages to and from the server. Because of this, we have chosen to focus our testing on the server, if the server has a 100% code coverage and all tests are satisfied, then we only need to ensure that the mobile application sends a correct message, to be confident in that the result is correct. The server layer consists of a class that handles all connection to the database, the goal is to achieve 100% method coverage on this class.

To avoid corrupting the data that is running on production, we made a script that makes an exact copy of the database to perform all of our tests on. An example of how the testing could corrupt the database is testing creation and deletion of a recipe. When the test is run, a recipe will be created to ensure that this functionality works. If for any reason this recipe could not be

deleted it would be possible for the users to see this recipe. The copying of the database could prove to be a bottleneck if the database were to get very big.

The unit tests are run using "PHPUnit" version 3.6.10 by Sebastian Bergmann[9], furthermore mutation tests are run using "judgedim/mutagenesis" [8]. The basic approach of using PHPUnit is to create a new file that includes the functions which should be tested. The new file must have a new class that extends `PHPUnit_Framework_TestCase`. All public methods in this new class, which are named with a prefix of `test`, will be included in the test. Inside each of these methods all kinds of assertions can be made, an example could be `$this->assertNotNull($var)`.

To demonstrate how we have performed the unit tests, the tests `testSetRecipe()` and `testGetRecipe()` have been included. These tests ensure that after a new recipe is written to the database, the exact same recipe can be retrieved again with no missing data.

```
1  /**
2   * @depends testSetUnit
3   * @depends testSetIngredient
4   * @covers DB::setRecipe
5   */
6  public function testSetRecipe($unit, $ingredient) {
7      $url = 'http://localhost/food/lib/recipebackend.php';
8
9      $fields = array(
10         'test'=>"true",
11         'recipeName'=>$this->recipeName,
12         'recipeDesc'=>$this->recipeDesc,
13         'group1'=>$this->groupName,
14         'group1_exchange1'=>"",
15         'group1_exchange1_mandatory'=>$this->firstExchangeMandatory,
16         'group1_exchange1_ingredient1'=>$ingredient,
17         'group1_exchange1_ingredient1_amount'=>$this->firstIngAmount,
18         'group1_exchange1_ingredient1_unit'=>$unit,
19         'step1'=>$this->firstStepDesc,
20         'step2'=>$this->secondStepDesc
21         );
22
23     $result = $this->sendCurl($url, $fields);
24     $this->assertNotNull($result);
25
26     $resultRecipeId = (int)strip_tags($result);
27     $this->assertGreaterThan(0, $resultRecipeId);
28
29     return $resultRecipeId;
30 }
```

Lines 2-3 The test depends on both `testSetUnit()` and `testSetIngredient()`, we need both ingredients and a unit of measurement to create a recipe.

Line 4 We specify that this test covers the `setRecipe()` method in the `DB` class. This is because the code coverage analysis tool does not properly detect coverage when using cURL[4].

Line 6 testSetRecipe() takes two parameters: the ID of the inserted unit of measurement from testSetUnit() and the ID from testSetIngredient() respectively.

Line 7 The $url variable is initialised to the value address of the file that handles creation of recipes. *recipebackend.php* is used by the recipe creator tool.

Lines 9-21 An array is initialised with key value pairs that represent the POST variables that is send to the given url.

Line 23 The request is send to the server with the given POST variables and the result is stored in $result.

Line 24 It is checked that the result is not null. If the result was null it would mean that something went wrong while inserting the data.

Lines 26-27 The ID of the newly inserted recipe must be greater than 0.

Line 29 Return the ID of the insert recipe. This ID is used by the testGetRecipe() method to check that we can extract the recipe correctly again.

```
1  /**
2   * @depends testSetRecipe
3   */
4  public function testGetRecipe($recipeId) {
5      $jsonResult = self::$DB->getRecipe($recipeId, $this->language, $this->
           metric);
6      $recipe = json_decode($jsonResult);
7
8      //Check that the recipe is equal to the one we inserted.
9      $this->assertEquals($recipeId, $recipe->id);
10     $this->assertEquals($this->recipeName, $recipe->name);
11     $this->assertEquals($this->recipeDesc, $recipe->desc);
12     $this->assertEquals($this->recipeImage, $recipe->image);
13     $this->assertEquals($this->groupName, $recipe->groups[0]->name);
14     $this->assertNotNull($recipe->groups[0]->order);
15     $this->assertEquals($this->ingredientName, $recipe->groups[0]->
           exchanges[0]->ingredients[0]->name);
16     $this->assertEquals($this->firstIngAmount, $recipe->groups[0]->exchanges
           [0]->ingredients[0]->amount);
17     $this->assertEquals($this->metricName, $recipe->groups[0]->exchanges
           [0]->ingredients[0]->unit);
18
19     $this->assertEquals(((bool)$this->firstExchangeMandatory), $recipe->
           groups[0]->exchanges[0]->mandatory);
20 }
```

Lines 2 The test depends on testSetRecipe().

Line 4 testGetRecipe() takes the ID that was returned from testSetRecipe() as a parameter.

Lines 5-6 The method to get a recipe is called from the DB class, and the JSON result is deserialised. This enables us to handle the result as a PHP object.

Line 7 The $url variable is initialised to the value address of the file that handles creation of recipes. *recipebackend.php* is used by the recipe creator tool.

Lines 9-19 Numerous assertions are made to ensure that the data in the fetched object is equal to the data we used when writing the recipe to the database.

Result

```
Time: 1 second, Memory: 3.75Mb
OK (16 tests, 45 assertions)
```

Listing 5.1: The result of the PHPUnit test.

The goal was to achieve 100% statement coverage of the DB class and as seen in Appendix C this was succeeded. Since all of the tests passes, we can be more confident that all of the functionality on the server works as intended. As long as the mobile application send the correct data of the correct format, a valid response should be sent back.

```
Score: 78.12%
(14 escaped on 64 mutants)
```

Listing 5.2: The result of the Mutagenesis test.

The tests have been run trough a mutation testing tool[8] to ensure that the tests works as intended. The mutation test will change certain aspects of the source code before running the tests. This includes changing strings to random strings, changing integers, flipping boolean statements, etc. If a test still passes after the source code has been altered, the test is considered defective. The mutation score is a good indicator for the quality of the code coverage since it would be easy to achieve a high code coverage if the assertions always passed regardless of input/output. The mutation testing tool gives a success rate of 78.12% which means that 78.12% of the mutants made the tests fail. It is a satisfactory success rate, but could be improved if we had more time.

Conclusion

6

The focus of this project was to make an application that would take advantage of the mobile platform and provide the users with relevant recipes based on specific ingredients. Furthermore, we wanted to support any user defined restrictions, like allergies. This is defined in our problem definition from Chapter 1:

> How can we take advantage of the mobile platform, in order to provide the user with relevant recipes based on specific ingredients, and taking any user defined restrictions, like allergies, into consideration.

In order to make the application easy for the users to use, a problem that had to be solved was how the users should navigate the application. We decided to use the navigation drawer and display it the first time the application is opened. We chose the navigation drawer because it is recommended to use when the application has more than three top level views in order to give the user an overview of the application.

The application itself consists of three pages, a page to search by ingredient, a page to search for recipes using free-text, and a favourite page.

When the user wants to search for recipes using ingredients, they click the search field to open a word cloud filled with already predefined ingredient suggestions. The user is able to click these and add them to the search. They are also able to search for ingredients using text. When the user types in the search field, they are given a list of suggestions based on the letters they have already typed. The user can either click an ingredient in the list to add it to the search or click the enter button on the keyboard to add the first suggestion in the list to the search. When the user is done entering ingredients they can perform a search by clicking the enter button while having an empty search field. This closes the word cloud and searches for recipes based on the ingredients that were selected.

The search provides the user with a list of recipes that either include all or some of the ingredients that they entered. The recipes are prioritised accordingly to our precedence function described in Section 2.5. The user can click a recipe from the list to open a page showing the full recipe. The users are able to favourite recipes by clicking the star in the top right corner. In order to favourite recipes they must be signed in through Google+. As long as they are signed in the users can always access the recipes through the favourite page. Through the favourite list they can also remove favourites by long clicking and click "remove" when prompted. They can also remove a favourite recipe by clicking the star in a favourited recipe.

The user is also able to search for recipes using free-text using wildcards such as *, + and "". In order to receive a result from the free-text search, the input text has to match either parts of the recipe name or part of the description of a recipe. A match on the recipe title is prioritised over a match

on the description, and a match on both title and description is prioritised over a match on the title.

Due to time constraints, we have not managed to fulfil all of our initial requirements as listed in Section 1.5. One of the major features that is lacking is the shopping list. It proved more complicated and time consuming to implement than first anticipated due to too much UI design on the application.

The intended purpose of the word cloud was to provide the user with relevant suggestions based on the ingredients they already entered to the search, however at the moment the suggestions provided are static and does not change. This basic implementation was a result of time constraints and how complicated it proved to implement something that gives relevant suggestions.

As the project progressed, we chose to focus on developing a working application with a few core features which meant that features such as search filters, sharing, persistency, unit conversion, and additional languages where deprioritised.

Based on the tests performed in Chapter 5, we can with confidence say that the server returns a correct and valid JSON format to the mobile application. Our black box test shows that the mobile application's functionalities works. We have not made any graphical user interface tests to ensure the design of our mobile application and website. We have managed to create an application which provides the users with the ability to search for recipes based on ingredients or using free-text.

Future Work

7

Due to time constraints in our project we have not managed to implement all functionalities. This chapter describes the features we did not have time to implement, but would like to implement in the future.

Shopping list We wanted to have a shopping list where ingredients could easily be added from recipes and also from the shopping list itself.

Start page The application needs some kind of a start page, right now the application opens in the ingredient search page. The start page can be a page displaying popular or featured recipes to inspire the user, but an automatic display of popular recipes might become a static page with the same recipes displayed, and featured recipes needs to be maintained. Another option for a start page can be a welcome message and a small guide or tutorial on how the application works and what it does.

Recipe filters The user should be able to specify some filters for search results. Examples are filters for vegans and allergies.

Sharing You cannot currently share anything in the application. We wanted the ability to share recipes and shopping lists. Sharing can happen through services like SMS, Facebook, or Google+.

Recipe cache Right now the recipes are downloaded each time they are opened. All the images are already automatically cached in the application, caching recipes also makes sense, since they are probably likely to opened multiple times, especially when they are favourited for later use.

Recipe license We need to display a license on all the recipes in the application. The application can currently display a license, but the server does not send it.

Conversion The database and the model currently contains a conversion constant for each unit, but the application does not contain the functionality to convert between units. Right now in the database decilitre is converted to ounces, but usually the imperial system deals with cups.

Recipe references We wanted recipes to be able to have references to other recipes, because ingredient groups might be big enough to be its own recipe. This is currently not possible.

Settings The application does not have a settings page. Settings could include: Metric or imperial standard option, filtering of recipes, and language.

Scaling of recipes Button to scale the ingredients of a recipe to more people.

Intelligent sort of ingredients Instead of having a static word cloud like we have now, we want the word cloud to update each time a user adds an ingredient to their search. We want the word cloud to update with suggestions based on the ingredients already added.

User created recipes In order to increase the collection of recipes, the users should be able submit their own recipes.

Localisation At the moment the application only supports English. We want to support more languages in the future in order to reach a wider audience.

Log interesting ingredient searches It could be very interesting to log what ingredients the users searches for the most. This could identify which ingredients the user search for and maybe help us improve our selection of recipes. It might help to improve the word cloud by identifying what ingredient combinations are mostly used.

Revoke Google account access The user should also have the option to revoke access, meaning the application would reset the granted access to the user's Google+ account. This option should be implemented in the settings menu.

Copyright clarification

A

Our question to the lawyers

Vi er en projektgruppe på Aalborg Universitet som er ved at udvikle en applikation med madopskrifter. I den forbindelse vil vi gerne vide hvordan ophavsrettighederne omfatter opskrifter.

Det er klart at man ikke må kopiere en beskrivelse af en opskrift fra en kogebog ord for ord, men må man bruge fremgangsmåden og listen af ingredienser til f.eks. en æggekage fra en kogebog?

I den amerikanske "copyright" lov er det ret tydeligt at den ikke beskytter en liste af ingredienser.
Link: http://www.copyright.gov/fls/fl122.html

Da der skal skrives en rapport om projektet, vil vi i den forbindelse høre om vi eventuelt må have lov til at citere dit svar i rapporten?

På forhånd tak,
Nicklas, Jesper, Jacob, Simon, Sam

Answer from Jens H. Bech

From: jb@amtmandstoften.dk
Date: 2014-04-01
Title: RE: Spørgsmål om ophavsret

Hej Nicklas

Jeg skal straks erkende, at jeg ikke beskæftiger mig meget med ophavsret. Jeg har fundet nedenstående på "Familieadvokaten.dk":

> *Kan man have ophavsret til madopskrifter?*
>
> *Det kan godt være, at det ikke lige falder ind under de mest gængse emner her på siden, men jeg har længe undret mig over, om madopskrifter er underlagt ophavsret.*
> *Kan man f.eks. skrive opskrifter af efter bøger og selv udgive de samme opskrifter - måske bare i en anden sammenhæng ?*
> *SVAR.*
> *Madopskrifter - eller kulinariske værker om du vil - er omfattet af ophavsretten (efter lovens §1), da der er tale om en slags faglitterære værker i skrift.*
> *Du har ikke ret "til at planke" - som det populært kaldes - andres opskrifter i blade og bøger for derefter at udgive dem i bogform.*
> *Men benytter du en offentliggjort opskrift til at udvikle din helt egen opskrift - fx med flere ingredienser og lidt anderledes mængdemål - har du frembragt et nyt kulinarisk væ rk, som du frit kan offentliggøre i fx en bog.*
> *Med venlig hilsen*
> *Erik Frodelund*
>
> *§1. Den, som frembringer et litterært eller kunstnerisk værk, har ophavsret til værket, hvad enten dette fremtræder som en i skrift eller tale udtrykt skønlitterær eller faglitterær fremstilling, som musikværk eller sceneværk, som filmværk eller fotografisk værk, som værk af billedkunst, bygningskunst eller brugskunst, eller det er kommet til udtryk på anden måde.*
> *Stk. 2. Kort samt tegninger og andre i grafisk eller plastisk form udførte værker af beskrivende art henregnes til litteræ re værker.*
> *Stk. 3. Værker i form af edb-programmer henregnes til litteræ re værker.*

Jeg kan endvidere se, at en dom fra 1983 (UfR 1983515H) slog fast, at en sammensætning af serier af mindre billeder (Mad fra A til Z) ikke blev anset for at være et litterært værk.

Håber I kan bruge noget af ovenstående eller skriv igen.

Med venlig hilsen

Jens H. Bech

Answer from Jørgen L. Steffesen

From: Jørgen Lindhardt Steffesen (Startvækst.dk)
Date: 2014-03-27
Title: RE: Ophavsret på opskrifter

Det er det man kalder et godt spørgsmål. Efter oplysningerne arbejder I reelt på at udgive en applikationsbaseret "kogebog", hvor det formentlig er tanken at man kan downloade en app der indeholder diverse madopskrifter. Så langt så godt. Det lyder som en glimrende idé. Næste skridt er så hvilke opskrifter brugeren kan vælge på app'en, og navnlig hvor de kommer fra. Hvis I eksempelvis kopierer en opskrift fra en af Claus Meyer's bøger om f.eks. ribbensburger - der iøvrigt er rigtig god - får i problemer med copyright. Dels må I ikke kopiere fra bøger uden tilladelse, og dels vil en ordret gengivelse af opskriften være en krænkelse af Meyer's rettigheder. Han anvender i den pågæ ldende burger bl.a. senneps-mayonaisse og æbler, hvormed han adskiller sig fra det der kan kaldes en standard-version. Til gengæld vil en gengivelse af standard-versionen ikke indeholde en krænkelse da det er alment kendt. Så længe I holder Jer til opskrifternes standardindhold og fremgangsmåde krænker I ikke nogen rettigheder.

Search by Ingredients Algorithm *B*

- The function `getIngredientSearchQuery()` returns a prepared statement which consists of the queries in Listing 3.5, Listing 3.6, Listing 3.7, Listing 3.8, and Listing 3.9.
- The function `getNotCoveredQuery()` returns a prepared statement of the query in Listing 3.10.
- The function `getIngredientsQuery()` returns a prepared statement of the query in Listing 3.11.

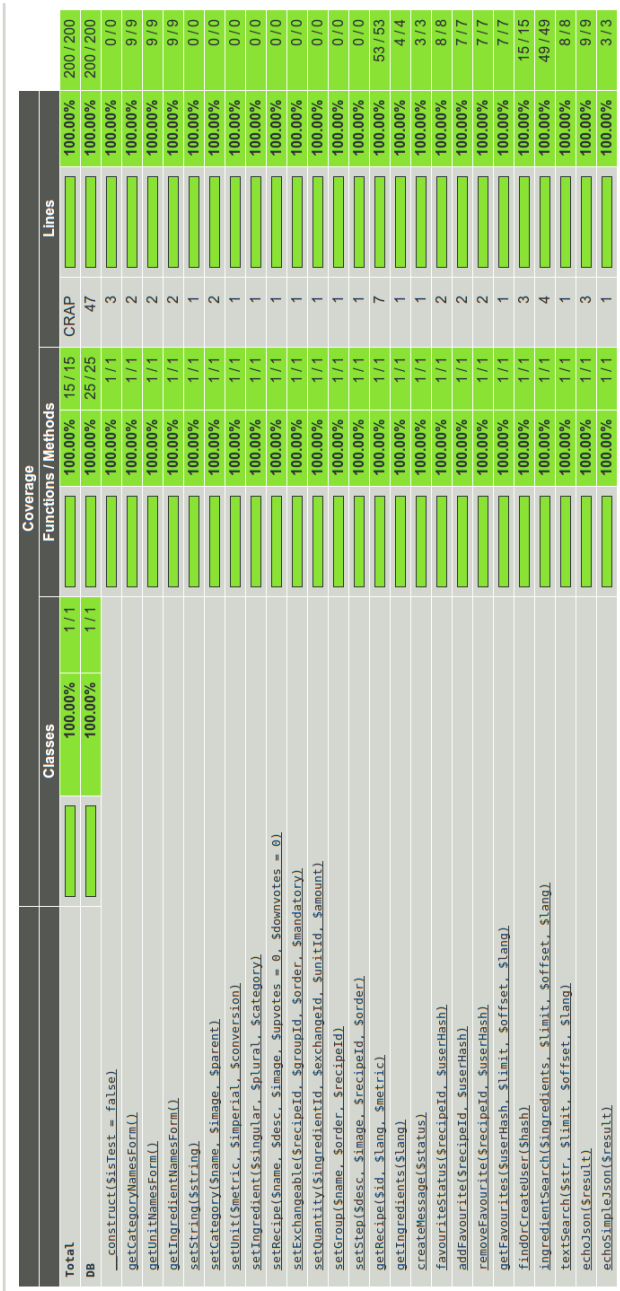
```
1  public function ingredientSearch($ingredients, $limit, $offset, $lang) {
2      // check whether the input is valid
3      if ($this->isNaturalNumber($limit) &&
4          $this->isNaturalNumber($offset) &&
5          $this->isValidLanguage($lang)) {
6          // ignored ingredients (water, salt, pepper)
7          $ignoredIngredients = '20,19,40';
8
9          // get the queries needed
10         $searchQuery =$this->getIngredientSearchQuery($ignoredIngredients,
               $limit, $offset, $lang);
11         $notCoveredQuery = $this->getNotCoveredQuery();
12         $missingQuery = $this->getIngredientsQuery($ignoredIngredients);
13
14         // prepare and bind the ingredients to the query. Bound 2 times since it is
               used in two sub-queries
15         $commaSeparatedIngredients = implode(',', $ingredients);
16         $searchQuery->bind_param('ss', $commaSeparatedIngredients,
               $commaSeparatedIngredients);
17
18         // execute and retrieve the output of the query
19         $searchQuery->execute();
20         $searchQuery->store_result();
21
22         // bind the output to $recipe
23         $recipe = new stdClass();
24         $searchQuery->bind_result($recipe->id, $recipe->name,
25             $recipe->description, $recipe->image);
26
27         // array to store the result
28         $result = array();
29
30         // for each recipe, append the missing ingredients
31         while($searchQuery->fetch()) {
32             // bind input, execute the query, retrieve the output, and bind it
33             $notCoveredQuery->bind_param('is', $recipe->id,
                   $commaSeparatedIngredients);
34             $notCoveredQuery->execute();
```

```
35              $notCoveredQuery->store_result();
36              $notCoveredQuery->bind_result($exchangeableId);
37
38              // array for missing ingredients for a recipe
39              $recipe->missing = array();
40
41              // find the missing ingredients for each exchangeable
42              while($notCoveredQuery->fetch()) {
43                  // bind input, execute the query, retrieve the output, and bind it
44                  $missingQuery->bind_param('i', $exchangeableId);
45                  $missingQuery->execute();
46                  $missingQuery->store_result();
47                  $missingQuery->bind_result($ingredientId);
48
49                  // if missing ingredients were found for this exchangeable, add
                        them to the result
50                  if ($missingQuery->num_rows > 0) {
51                      // array for missing ingredients for an exchangeable
52                      $missingIngredients = array();
53
54                      // add missing ingredients to the array
55                      while($missingQuery->fetch()) {
56                          $missingIngredients[] = $ingredientId;
57                      }
58
59                      // add the missing ingredients for this exchangeable to the
                            result
60                      $recipe->missing[] = $missingIngredients;
61                  }
62
63                  // free the resources
64                  $missingQuery->free_result();
65              }
66
67              // remove duplicates and reindex array
68              $recipe->missing = array_merge(array_unique($recipe->missing,
                        SORT_REGULAR));
69
70               // we need to make a deep copy since $recipe is an array of refs
71              $result[] = $this->deepCopy($recipe);
72
73              // free the resources
74              $notCoveredQuery->free_result();
75          }
76
77          $searchQuery->close();
78
79          return $this->echoSimpleJson($result);
80      }
81  }
```

Listing B.1: Search by ingredients.

Code Coverage

C



Legend: executed not executed dead code

	Coverage							
	Classes		Functions / Methods			Lines		
Total	100.00%	1 / 1	100.00%	15 / 15	CRAP	100.00%	200 / 200	
DB	100.00%	1 / 1	100.00%	25 / 25	47	100.00%	200 / 200	
__construct($isTest = false)			100.00%	1 / 1	3	100.00%	0 / 0	
getCategoryNamesForm()			100.00%	1 / 1	2	100.00%	9 / 9	
getUnitNamesForm()			100.00%	1 / 1	2	100.00%	9 / 9	
getIngredientNamesForm()			100.00%	1 / 1	2	100.00%	9 / 9	
setString($string)			100.00%	1 / 1	1	100.00%	0 / 0	
setCategory($name, $image, $parent)			100.00%	1 / 1	2	100.00%	0 / 0	
setUnit($metric, $imperial, $conversion)			100.00%	1 / 1	1	100.00%	0 / 0	
setIngredient($singular, $plural, $category)			100.00%	1 / 1	1	100.00%	0 / 0	
setRecipe($name, $desc, $image, $upvotes = 0, $downvotes = 0)			100.00%	1 / 1	1	100.00%	0 / 0	
setExchangeable($recipeId, $groupId, $order, $mandatory)			100.00%	1 / 1	1	100.00%	0 / 0	
setQuantity($ingredientId, $exchangeId, $unitId, $amount)			100.00%	1 / 1	1	100.00%	0 / 0	
setGroup($name, $order, $recipeId)			100.00%	1 / 1	1	100.00%	0 / 0	
setStep($desc, $image, $recipeId, $order)			100.00%	1 / 1	1	100.00%	0 / 0	
getRecipe($id, $lang, $metric)			100.00%	1 / 1	7	100.00%	53 / 53	
getIngredients($lang)			100.00%	1 / 1	1	100.00%	4 / 4	
createMessage($status)			100.00%	1 / 1	1	100.00%	3 / 3	
favouriteStatus($recipeId, $userHash)			100.00%	1 / 1	2	100.00%	8 / 8	
addFavourite($recipeId, $userHash)			100.00%	1 / 1	2	100.00%	7 / 7	
removeFavourite($recipeId, $userHash)			100.00%	1 / 1	2	100.00%	7 / 7	
getFavourites($userHash, $limit, $offset, $lang)			100.00%	1 / 1	1	100.00%	7 / 7	
findOrCreateUser($hash)			100.00%	1 / 1	3	100.00%	15 / 15	
ingredientSearch($ingredients, $limit, $offset, $lang)			100.00%	1 / 1	4	100.00%	49 / 49	
textSearch($str, $limit, $offset, $lang)			100.00%	1 / 1	1	100.00%	8 / 8	
echoJson($result)			100.00%	1 / 1	3	100.00%	9 / 9	
echoSimpleJson($result)			100.00%	1 / 1	1	100.00%	3 / 3	

Figure C.1: Code coverage analysis.

List of Figures

List of Listings

Bibliography

[1] Allthecooks's google play. https://play.google.com/store/apps/details?id=com.mufumbo.android.recipe.search.

[2] Bigoven's google play. https://play.google.com/store/apps/details?id=com.bigoven.android.

[3] Creative commons. http://creativecommons.org/.

[4] curl. http://www.php.net/manual/en/intro.curl.php.

[5] Free cultural works. http://freedomdefined.org/.

[6] Android guidelines - app structure, . https://developer.android.com/design/patterns/app-structure.html.

[7] Android guidelines - navigation drawer, . https://developer.android.com/design/patterns/navigation-drawer.html.

[8] Mutagenesis. https://github.com/judgedim/mutagenesis.

[9] Phpunit. http://phpunit.de/.

[10] Supercook. http://supercook.com/.

[11] Fulltext index overview, May 2014. https://mariadb.com/kb/en/fulltext-index-overview/.

[12] Mariadb vs. mysql, March 2014. https://mariadb.com/kb/en/mariadb-versus-mysql-features/.

[13] Central Intelligence Agency. The world factbook, weights and measures, 2013. https://www.cia.gov/library/publications/the-world-factbook/appendix/appendix-g.html.

[14] Android. Action bar, May 2014. http://developer.android.com/guide/topics/ui/actionbar.html.

[15] Android. Activity, May 2014. http://developer.android.com/reference/android/app/Activity.html.

[16] Android. Fragments, May 2014. http://developer.android.com/guide/components/fragments.html.

[17] Google Developers. Google i/o 2013 - google+ sign-in for android developers, May 2013. http://youtu.be/_KBHf1EODuk?t=4m53s.

[18] Google. GoogleApiClient - life cycle, Marts 2013. http://www.riskcompletefailure.com/2013/03/common-problems-with-google-sign-in-on.html.

[19] Google. GoogleApiClient - API documentation, May 2014. http://developer.android.com/reference/com/google/android/gms/common/api/GoogleApiClient.html.

[20] Business Insider. One in every 5 people in the world own a smartphone, December 2013. http://www.businessinsider.com/smartphone-and-tablet-penetration-2013-10.

[21] Samantha Murphy Kelly. Mobile devices will outnumber people by the end of the year, February 2013. http://mashable.com/2013/02/06/mobile-growth/.

[22] W3C Richard Ishida. Internationalization, text size in translation, 2011. http://www.w3.org/International/articles/article-text-size.en.