Buy Locally – get in touch with producers

Jan Lewandowski

Contents List

Analysis

Problem Identification

Creating an application where people can search and find local handmade producers and farmers to buy things straight from the source and support local sellers. The concept is that producers can set up an account, mark their location on the map and choose what things they make and want to sell. Customers will set up an account, search by the location they are interested in and the things they want to buy. The customers will also be able to give reviews.

In order to create my application, I will use **abstraction** to remove unnecessary detail and simplify producers and users to their essential characteristics such as location, type of products and name and email address. These characteristics can identify each producer accurately. Without abstraction, it would be too complex as each producer has a farm of a different size, shape, they are different ages and so on. Taking all of these characteristics into account would reduce the efficiency severely and overcomplicate the program, as such irrelevant attributes would have to be incorporated and implemented. Abstraction will enable me to make the problem easier, more straight-forward to solve, and more understandable. **Logic, branches, and loops** will be used to respond to the user's input, the program has to react upon the user's clicks and actions. For example, if the user clicks on a button to sign up, the program should verify their details and then sign them in if the details are checked to be correct. Another technique I will use is **decomposition**. I will divide the problem into smaller ones, to make my work organised, like the structure of a customer and of a producer. Other decomposed parts of the problem are also the login, system, and reviews. I will make the program **components reusable** because it will make coding and designing easier. Once I implement a class and test it, I will be able to use it throughout all succeeding development. The code will also be more organised and neat. Also, I am going to specify **preconditions** of functions. This will enable me to avoid many errors while writing the code and probably prevent some possible bugs later on. Handling **input and output** well will be as important. **Caching** might be crucial for the correctness and effectiveness of my application as well. It could be used to remember the user's recent choices. Also, the user's last known location will be stored in cache so that I can access it quickly and display it on the map. Thanks to cache memory, I will be able to retrieve this data very quickly.

Therefore, the problem is amenable to a computational approach because creating the solution will involve storing a lot of data about its users to allow the program to display all the registered producers in the app. Doing this by hand would be difficult and time-consuming as large amounts of data such as information on a large range of producers, the types of products and the lists of products sold, as well as each users' favourites. After this data is collected, it will need to be stored and retrieved frequently. If this was done by paper and stored in a real-life office cabinet, trying to frequently access different pieces of data will be very tedious. Hence, this can be implemented most efficiently through a database system.

Additionally, my program will involve a dynamic map with real time data added to it, such as the current user's location and markers of all registered producers. Doing this by hand would

be impossible, to get the user's current location and display it on the map that they can interact with—move around, magnify. Contrastingly, a computer can download this data from the database, create a map and add the necessary data onto it. Hence, my problem is amenable to a computational solution.

Stakeholders

My stakeholders will be people who do not want to do shopping in big markets and shopping centres. These are the people who care about their health and would like to support their local community. My main end-users and therefore testers will be two of my friends - Daniel Uko from England and Szymon Jończak Lis from Poland.

Daniel Uko and his family would like to buy more ecological and healthier food straight from the source such as vegetables, fruit, meat, and dairy. They prefer to pay a little more and know the origin of their supplies than to just go to a supermarket. They are not sure whereabouts in their area are different producers offering products they need. Also, they don't have the time to do research on all the existing farmers nearby them. My application will be the perfect solution as it will show them all registered producers in one place together with the type of products they sell. This allows them to quickly decide which producers to contact and get in touch with. Daniel and his family will use the application to discover local producers close to them. They will be able to sort the producers by the type of food they want to buy at the moment, and when they choose a farmer they like, they can immediately contact them all within the app. Daniel's parents are elderly, so they really value the fact that they would be able to manage and contact all local producers in one place.

The second stakeholder - my friend from Poland, Szymon Jończak Lis - is a vegetarian and lives in the countryside. He and his mother would like to get in touch with farmers selling fruit and vegetables in their area to support locals and know the origin of their food. However, they don't have the time to look for farmers and seek all the necessary information to start buying from them. My application would allow them to see all producers in seconds, together with their contact information and what they sell. This will save my friend and his mother a lot of time and hassle to be able to start contacting the producer they need in one place easily. They are busy people, so my app will answer their needs with one program to discover, manage and contact local producers. Thanks to the application they will be able to supply all necessary fruit and vegetables and know directly where they come from, so that they can improve their health and continue the vegetarian diet.

Existing solutions

- BigBarn

BigBarn is the biggest local food website in the UK. It offers a definitive database of producers who want to sell directly. The user interface is pretty clean and straight-forward, but the main page is full of information which not everybody might need. Fortunately, the categories at the top make navigation a bit easier. There are multiple categories, such as Home, the Map, Marketplace. There is also an option to look for producers by postcode and by products, which is really handy. The map is really well-made, with the possibility of filtering the icons by the type of product.

Advantages of the website:

- clear, tidy UI
- ability to search by product or by localization
- icons on the map symbolise the product that is being sold
- food is divided into categories like bakery, dairy, fruit etc.

Disadvantages:

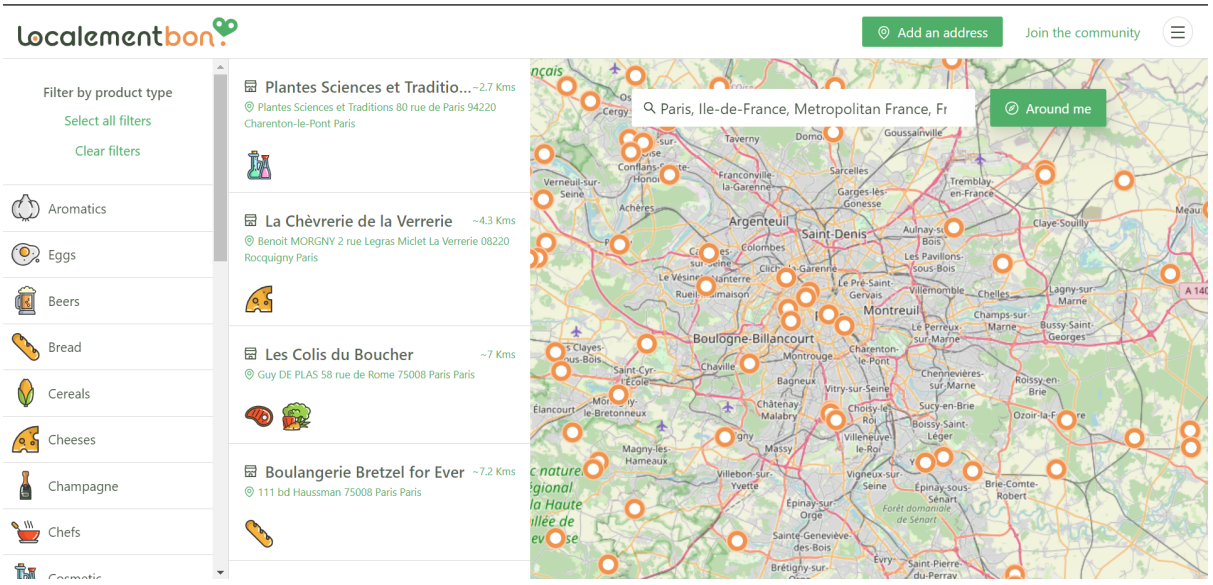
- main page appears a bit confusing because it contains many segments which are not necessary in my opinion, such as news, recipes, deals
- cannot give ratings in a star system
- the icons of products are all green and white, which makes them really boring and hard to distinguish

What would I use and I would not use?

- the search box design, searching by product or by location
- the way food is categorised, because it will really make the searching clearer
- icons on the map, although I do not like the style of them, and I would use similar ones, I really like the idea of putting icons on the markers on the map and I will implement it
- I would not make the main page so overpacked
- Unlike on this website, I would also implement a review system

To conclude, this website is pretty resourceful and offers many important features. I would be inspired by the way the map looks and the searching system. Also, the structure of food categories is well-designed. Unlike the website, I would make a review system where every producer can get from 0 to 5 stars, rather than only long-written reviews. I would also certainly change the style of icons because they are all green and look similar to each other. The main page of my application will also be different, because I think on the main page there should only be the key features, such as search box and login segment and other information should be in different categories because not everybody wants to see them all.

- Localement Bon



Localement Bon is a website that enables customers and producers to set up an account and buy or sell products from many various categories around the chosen location. Unfortunately the only places on that website are from France and Germany. It has less features, but has a good-looking and encouraging login system. The main page only contains the key segments, the search bar, the join-in banner and a list of some new producers, which makes it extremely comfortable to navigate and use.

Advantages:

- straight to the point, no unnecessary information
- very good product categorization
- extremely easy to join and set up an account
- you can add producers to your favourites

Disadvantages:

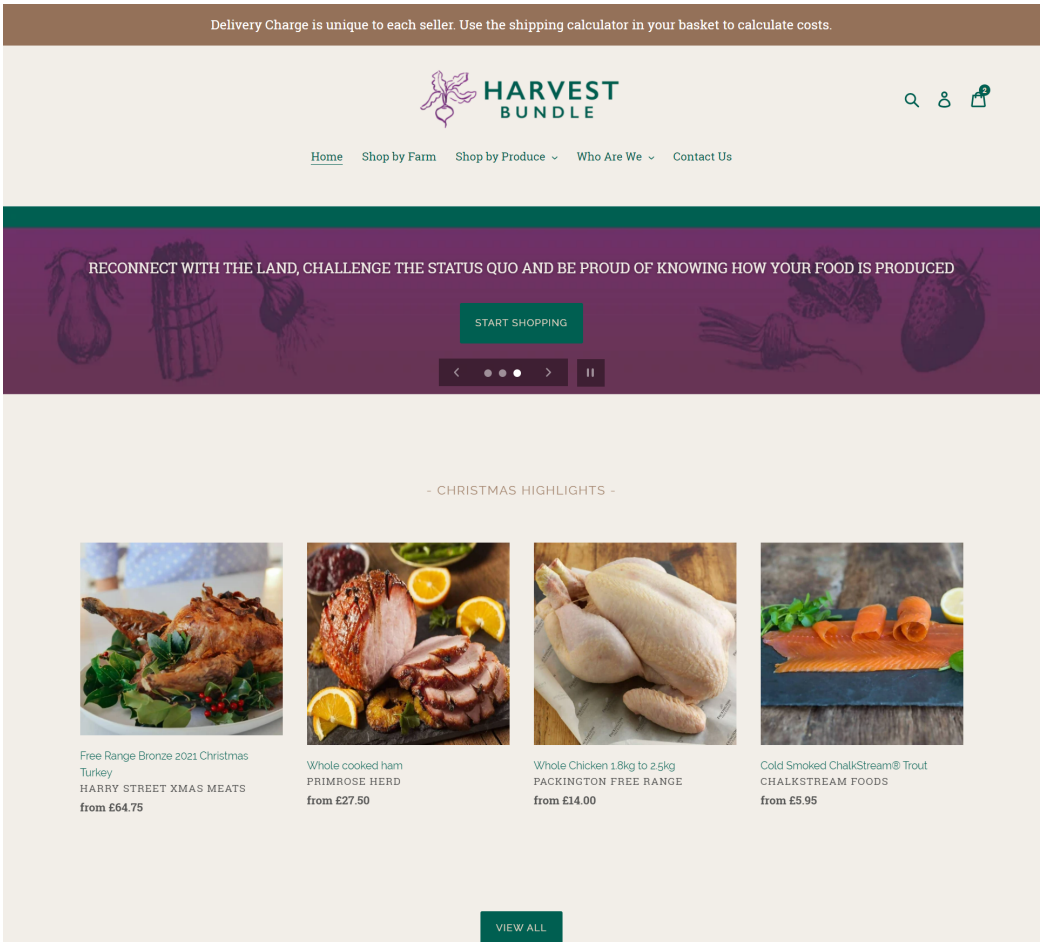
- cannot give reviews
- icons on the map are not informative
- cannot search by producer or the product

What would I use and I would not use?

- I would use the look of the main menu, because it is very clear
- unlike on this website, I would also implement a review system because it will give the user a quick insight of the seller
- I would use the option of adding producers to favourites which will be very convenient for the users, shortens the time of searching, the users do not have to always look for their favourite seller by his name
- I would not use the style of the map, because the icons are very uninformative
- I would add on option to search by product, which is absent on this website

Overall, this website is pretty minimalistic and the design does its job, but it lacks some essential features. I would probably use the categories of the products because they give much information. The style of the icons is really soft and low-poly, which I find pleasant to watch. Also, the login system is very clear. But I would not use the icons on the map because they are just dots which all are the same. I would implement a review system, which is absent here.

- Harvest Bundle



Harvest Bundle is a website that connects over 50 farmers from all over the UK. The main categories of available products are meat, fish, fruit and vegetables, dairy and eggs. The website UI is really tidy and easy to navigate. However, unlike the previous examples, here there is no option to mark your location and see producers that are situated in your proximity. This website is more like an online shop. You can search by product or producer and add them to the basket. At checkout, the cost of delivery from each farmer is calculated.

Advantages:

- clear main page, can quickly start shopping
- when browsing through a farmer's page can see all their products and sort them in categories or alphabetically
- can send queries to sellers right from their page

Disadvantages:

- cannot browse the producers on a map
- there is no review system
- cannot easily join as a producer
- cannot search by localisation

What would I use and I would not use?

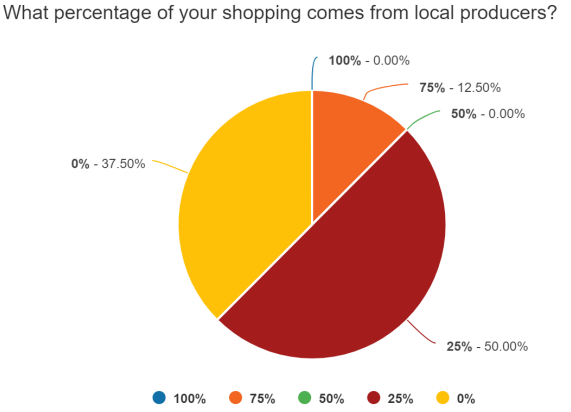
- I do not think I would use the look of the main page, although the customers can very quickly begin the shopping, which is a good thing, there is no map, and it is not clear how to set up an account
- I would use the way the pages of the farmers work, because they are really informative and well-designed, can write a message and browse by product
- I certainly do not want to eliminate a map completely and only let for deliveries, I think this would be an utterly bad decision as it would block the flexibility and ruin the experience as some people probably would like to see the farms themselves and if they live close they do not want to pay for shipping
- I would certainly make an option to join as a producer, unlike this website

Overall, although the website has some conceptually bad decisions in my opinion which I would not incorporate in my application, it is pretty well-designed with some interesting solutions, such a good-looking producer page or the categorisation. But it lacks some essential features which I want to incorporate in my application, such as a map with marked locations of customers or a straightforward login system.

Feedback from survey

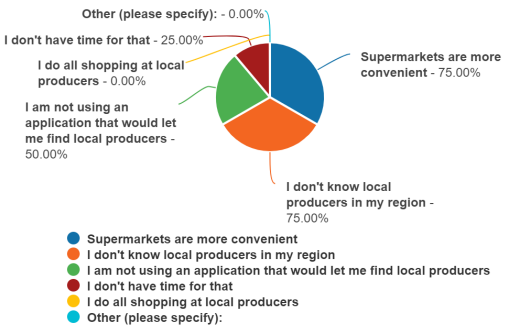
My survey directed to stakeholders contained questions that will help me establish some of the crucial features and the general nature of their shopping practices.

At first, I asked a very general question about how often people go shopping. The results show that nobody goes shopping less than once a week. Most people go shopping a few times a week, then about once a week. However, only one-eighth of people go shopping every day.



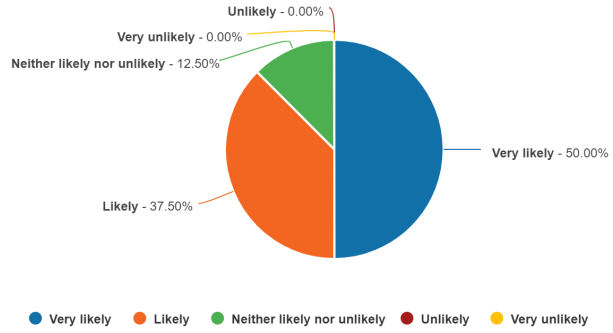
The next question is crucial for the purpose of my application—what part of shopping is from local producers. The profound majority of people's food is not from local producers (25% or none). A very small group of responders however do not follow the general tendency.

If You are not doing all your shopping at local producers, what are the reasons?



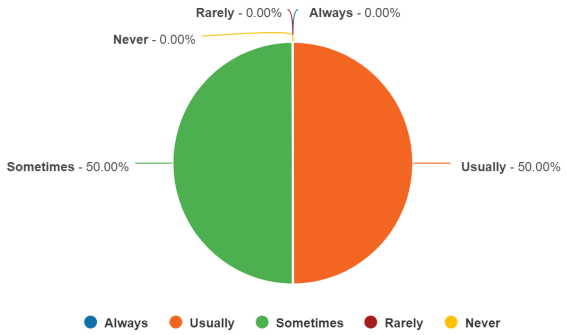
This question directly follows the previous one – why do not buy from local producers. The majority indicated that the reasons are – "I don't know local producers", "Markets are more convenient", and crucial "I am not using an app to find them". A small portion also says that they do not have time for that.

Would You do more shopping at local producers if you used an application that would allow You to find them more easily?



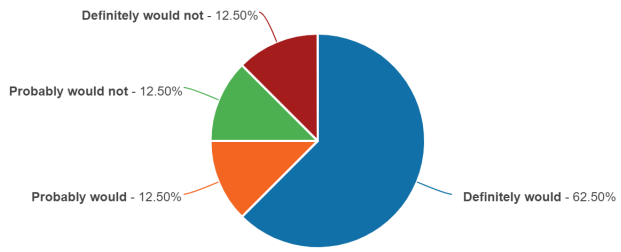
This chart clearly shows the need for my application – "would you do more shopping from local producers if my app existed". The complete majority says that it is likely or very likely. A small part is neutral, but nobody thinks they would not use my app.

Do You buy the local, healthier food even if it is more expensive than in supermarkets?



A follow-up question, which shows that people buy the more expensive ecological food sometimes or usually.

Do You think You would use a review system to rate the producers within the application?



The answers to this question justify the need of a rating and review system. Only a tiny part think it is not necessary.

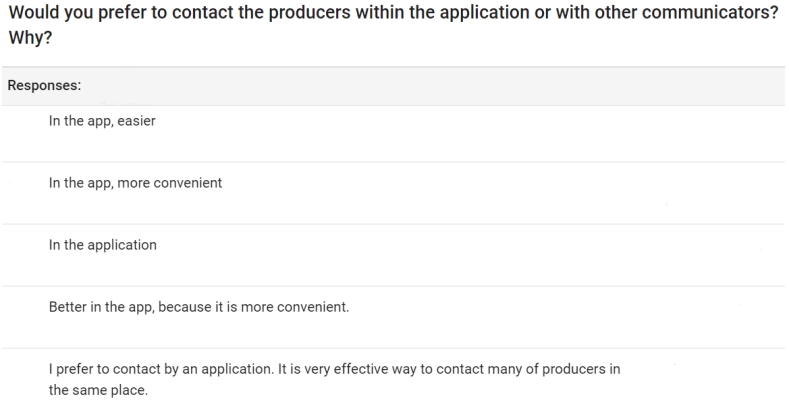
7	Style no2, because it is more clear
4	Style 2 because I am used to this kind of maps
3	Map 2 is better because it indicates the marked points
6	Style no2, better icons and colours

Next question was asking which of the maps from existing solutions – number 1 - Localement Bon and number 2 - BigBarn. Almost all the respondents said that they like the second map more. Here are some of the vital answers.

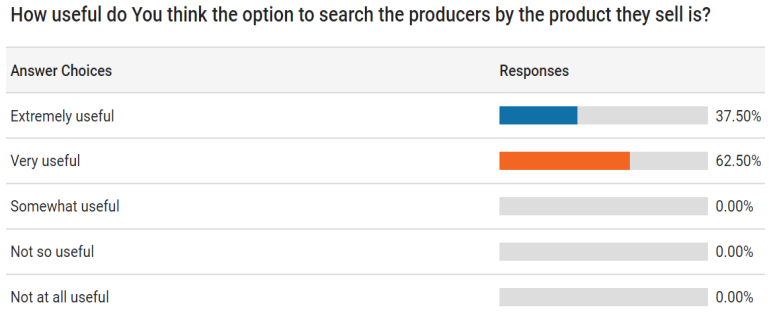
Please assign every category an indicator of how much interested are You in buying given things from a local producer

Answer Choices	Not at all interested	Not so interested	Somewhat interested	Very interested	Extremely interested
Vegetables	0.00%	0.00%	12.50%	25.00%	62.50%
Fruit	0.00%	0.00%	0.00%	50.00%	50.00%
Meat	0.00%	12.50%	50.00%	37.50%	0.00%
Dairy	0.00%	12.50%	50.00%	37.50%	0.00%
Poultry	0.00%	62.50%	25.00%	0.00%	12.50%
Honey	0.00%	0.00%	50.00%	12.50%	37.50%
Fish	0.00%	37.50%	25.00%	25.00%	12.50%
Eggs	0.00%	12.50%	37.50%	25.00%	25.00%
Bakery	0.00%	12.50%	37.50%	12.50%	37.50%
Alcohol	25.00%	25.00%	50.00%	0.00%	0.00%
Jams	0.00%	25.00%	50.00%	25.00%	0.00%
Oils	0.00%	62.50%	12.50%	12.50%	12.50%
Sweets	12.50%	25.00%	25.00%	37.50%	0.00%
Cosmetics	0.00%	25.00%	75.00%	0.00%	0.00%

The next question was not as oriented on the features, it was more focused on examining the general needs of my stakeholder regarding the products they would like to buy.



This question was about one of the important features – the chatting system with the producers. All responders believe that it should be within the application, and here are some answers.



The last question of my survey also asked about one of the crucial features – searching by a product you are looking for. Everybody thinks it is useful or very useful.

Project requirements

Requirement		Essential feature	Solves	Success Criteria	Justification	How will I test this
		Search box	**Search effectively**			
1	1.1	Search by the postcode	Search by location	Making a working search box to enter a postcode and check if the format is correct	This is the most common method of searching, as the users want to discover producers close to them or in a place they know exactly by the postcode	I will test this by selecting the search by postcode option and then inputting a postcode and seeing if the correct producers were found
	1.2	Search by producer	Search by a farmer name	Creating a search box to search by the name of a producer and	Sometimes customers want to search by the name of the farm, not knowing the exact	I will test this by selecting the search by producer option and then inputting a

				sorting alphabetically	location	name and seeing if the correct producers were found
	1.3	Search by products	Search by the type of food	Creating a search box that enables searching by the products and their categories	It will also be convenient to search by the product the customer wants to buy, which is also proven by survey	I will test this by selecting the search by products option and then inputting a product name and seeing if the correct products and producers associated with them were found
		Map	**Map handling**			
2	2.1	Meaningful icons	Clarity of the map	Thoughtful, well-designed icons which are meaningful	Icons showing what product is sold in a particular location will make the map much more comprehensive and easy to use, also supported by survey	My stakeholders and their answers to the questionnaire will test the design of the icons, how the users like them and if they are clear and easy to understand
	2.2	A box to put the postcode	Find the location the customer is interested in	Giving the user ability to locate himself on the map	The ability to put the postcode of the desired location will make using the map less tedious, as the user will be able to move the map focus to where they desire	I will test this by inputting a postcode into the field and seeing if the map move correctly and accordingly to the postcode given
	2.3	Using conventional colours	Make the map pleasing to the eye	Using matching colours, also not very bright or just too saturated, colours which will be pleasant to look at for the users	Using standard colours, similar to Google maps will make using the map more enjoyable and familiar for the user, because the map is a place where the users will spend a lot of time and also supported by	My stakeholders and their answers to the questionnaire will test the adequacy of the colours used and their saturation, as well as their familiarity and the feel

					survey	
		Log in screen	**Secure log in to the application**			
3	3.1	Register button	Enabling the user to set up an account	Creating a clear labelled button to register as a customer	The user must have an option to register with a unique username and password, as this is the essential method of authenticating users	I will test this by using the register page and trying to create an account, both using correct and erroneous data to see if the data is correctly validated and stored in the database
	3.2	"Register as a producer" button	Enabling the user which is a farmer to set up an account	Creating a clear labelled button to register as a producer	I want to give every farmer the opportunity to register in my app, so that the users can check them out, contact them and eventually buy products from them	I will test this by trying to register as a producer to see if the account is correctly created and stored in the database
	3.3	Password box	Requirement to enter the password	Creating a box taking in a password and checking if it meets the requirements	The password that the user enters must be secure, containing big letters and numbers to make it more secure and less prone to hacking	I will test this by inputting a string as a password and seeing if it has been correctly validated according to the password rules
		Database of customers	**Handling users profiles**			
4	4.1	Login details fields in the database	Remembering the details used when logging into the application	Creating fields in the database - username, email, password (hashed)	When user logs in we must check if the email exists and if the password is correct, so that only the person with the right credentials can enter the system	I will test this by trying to log in to a previously created account as well as inputting incorrect details to see if the password is stored and checked correctly

	4.2	Storing user favourites	As a customer you can mark producers as favourite and access them more quickly	Creating a field in the database - favourites	It will make finding and contacting producers which the user likes the most much faster. Next time a user wants to access a producer that they frequently check out they can do so through the main page	I will test this by adding some producers to favourites and going back to the home page to see if they are displayed
	4.3	Storing information about all the producers	Remembering all the locations, products, names, contact info of the producers	Creating the sufficient fields in the producers database	I must have a database containing all the producers to then quickly display their products, contact info	I will test this by checking out producers' pages that I have previously created to see if the details are stored and displayed correctly
		Chatting system within the app	**Communication with producers**		Answers in the survey confirm the need of the chatting system	
5	5.1	A "start chatting" button in producer page	Quick and convenient communication right from the producers page	Implementing a button that will redirect the user to the chatting segment to the correct conversation	It is a solution for those who find a producer they like and want to contact them on the spot, without having to user other communication software	I will test this by trying to start a new chat with a producer and seeing if I can send and receive messages in real-time
	5.2	Chatting page itself	Displaying the current conversations of the users as well as producers, probably last message and a profile picture	Creating a "feed" of messages with the newest ones on top	Enables the users to go into whatever chat they started and want to continue, as well as managing multiple chats at the same time easily	I will test this by going to the chats tab and seeing if all the current chats and listed there with the correct data, the list is scrollable and displayed neatly

	5.3	Notifying the user when they get a new message	Enables the user to see the messages even when they are not currently using the application	Getting permission to push notifications to the user's phone and displaying them as a notification	The user should know if they get a message even if they do not check the chats in the application	I will test this by sending a message to the producer and seeing if they get a notification of the incoming message on their phone

Limitations of the project

Limitation	Justification
Only available for Android devices	My development environment is Android Studio and the language I am using is Java, so the application will only work for Android, and it will not work on Apple devices. The development for Apple uses a completely different programming language. I don't have any experience with it and it is much more complex. I cannot use the same codebase for Apple phones. I am also for example using SQLite, to store all databases, which is only available on Android phones.
I am just a single developer	As a single developer, I will not be able to implement every single feature which the stakeholders might want to include in the application. As the single and only code-writer, I might not be able to implement every functionality to the highest degree of completion and cleanliness, because I am sometimes using mental shortcuts, which might not be easily understandable by others.
Time	I am not a full-time developer, I will not be able to implement every possible feature because of that. The time frame for the project is limited and I have to finish the development before the deadline.
Funds	Because I am not investing funds in the application the design and icons might not be the best possible and every feature tested perfectly, but of course I will try to test everything thoroughly. The server throughput and response time might not be the best possible.

Suitable approaches

One of the approaches I could use is to create an app for iOS using an interface such as Xcode. This approach would allow me to deploy the app for Apple devices. BigBarn, which is one of the existing solutions I researched, has a style similar to Apple design. I could get inspired by BigBarn while developing the application for iOS, using similar map design or style of icons. This would help me appeal to current iOS users. However, in my case this would not be convenient, as I myself do not own an Apple device and neither do my stakeholders. On top of that, the language used for iOS development is Swift, which I do not have any experience in and it would be pretty difficult for me to start iOS development from scratch and no experience. Creating an application for iOS would also vastly limit my end-users pool, as Android phones take up almost three quarters of the mobile market share worldwide. My end-users, both Daniel and Szymon and their families use Android devices. Therefore, I will not use this approach.

Another approach I could use is Python programming language with the Kivy library. This library would enable me to implement an application which can run on both Android and iOS. However, this library cannot utilise native Android or iOS functionalities, such as using Google Maps inside the program in Android, which for me is one of the crucial features, as my application is based on being able to find new producers on the map. This functionality is also wanted by the stakeholders, which can be seen in the survey answers, the users are very familiar with the Google Maps look and composition. In addition, my end-users value accessibility and the native functionalities of Android, which are not all available if using Kivy. That is why I am not going to use this approach.

The next approach is by using Android Studio and writing the code in Java. This approach will grant the application native performance on Android phones and give me access to the native Android libraries, modules and functionalities. I myself have an Android phone, which will make testing the app much more convenient and quicker. I will also be able to use native Google Maps map activity inside the application, which is desired by the stakeholders. Most of the websites I researched as the existing solutions have more of an Android design style. I will be able to inspire my app design to the existing solutions, such as Localement Bon or Harvest Bundle. Thanks to that, my application will appeal to Android users by using a similar style. My end-users use Android devices and performance, for example loading times is important for them, so Android Studio will be the perfect approach, as it will allow me to use all native Android features and ensure the performance is top notch. I am pretty confident in Java, so I think this is the best approach for my project and I will use this approach.

Hardware requirements

Requirement	Justification
GPS connection	This is necessary to locate the user on the map

2 GB of RAM	This will enable the app to run smoothly and also to store cache data to quickly retrieve information
Free disc space – around 50mb	This space is necessary to install the application on the user's device and use it properly with all functionalities

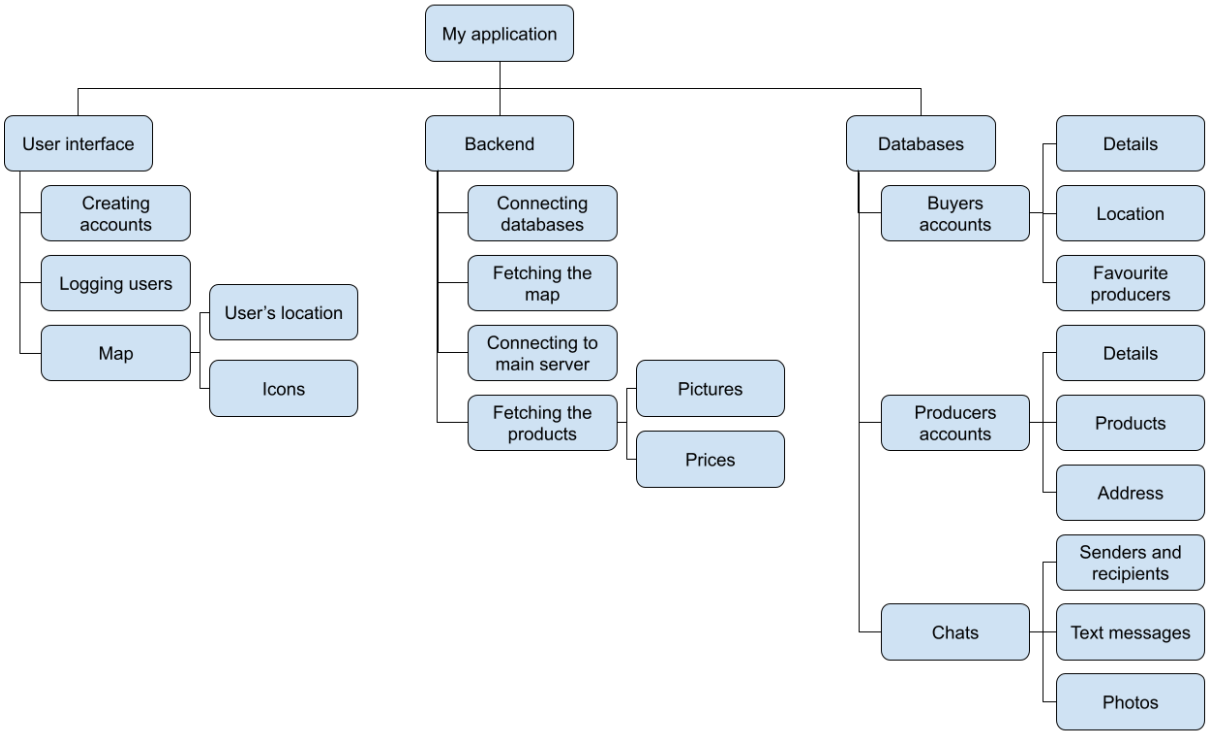
Software requirements

Requirement	Justification
Operating system – Android 5 or higher	The app will work on almost all versions of android, only the very old ones are not supported, and I will not be able to make this work on them
Android Studio	This is the environment I am making my application in, so I need it installed on my laptop

Design

In the design stage I am going to split into smaller sections which then will be combined to form the final application. This will allow me to write clearer code and better manage the application as a whole. These sections will be the user interface, backend, and databases. As the UI is a major section, I will split it into smaller, independent parts. They will include the login screen, register an account screen, the main map page and the chats page. Decomposition will enable me to work on the separate sections independently, and also testing them will be much easier. I will show the structure and the further decomposition on a top-down diagram.

Decomposition diagram



I decomposed the application into three main branches - user interface, backend and databases. This division will allow me to focus on the different vital parts of the application separately.

Databases are a vital part of my application. They will be very frequently used throughout the program, when registering an account, logging in or storing newly created producers. Therefore, I made them split them onto their separate parts, and I will make sure they work correctly, as other components depend on them.

The next segment is the backend, which is the connection between databases and the user interface. It is a separate section, because I will ask the databases very frequently, and it will take care of querying the databases and returning the data in the correct format.

User interface is another main part of the project, it is what the user will be able to see at all times. It is split into three branches - accounts handling, logging and the map itself with the user's location and all the icons on the map, which indicate all the producers in the current range of the map opened. Backend is divided into four subproblems. First is to connect all databases, user’s accounts, producers locations and products they sell, chats. Second is fetching the map using the Google maps API and also reading all the producers’ locations from the database. Then another crucial subproblem is connection to the main server of the application to keep every user connected at all times. At last, when the user browses the producer’s page, it is needed to fetch all the products the given producer sells, their pictures and prices. The last of the main branches are databases, which are divided into accounts - buyers and producers - and the chats. Every buyer has his details, location and favourite

producers. Every producer has his details, products, and the address. Chats consist of the messages themselves, also possible photos and who sends them and receives them.

This division into subproblems will enable me to work on them independently and also test them independently. This will assure that each part is working, and then I will be able to connect all the parts together to form the final application. The program implemented in this way will be robust, easy to understand and straightforward to develop in the future and maintain.

System objectives

Primary requirements

User interface

- Side bar to navigate through the main map page, chats, favourite producers, searching
- Map with indicated producers
- In producer's page display their description and products
- Enable moving the map
- Display a small producer's page when clicking on an icon
- Display a button to see the full producer's page
- Display searching box
- Displaying the rating of a producer

Processes

- Update databases when buyers and producers register an account
- Check if logging details are correct
- Algorithm to return based on the searched word - by producer name, product
- Bring the map the location user inputs
- Detect the location if the user consents to it
- Calculate the rating based on reviews

Inputs

- The user's location - from gps or given manually
- Select a producer from the map
- Select a producer from chats list
- Clicking sidebar button to navigate
- Enable giving reviews

Outputs

- Put the producers marks on the map
- Send chats in real time

- Output the products a given producer offers

Secondary requirements

- Being able to add “friends” and see their favourite producers
- Being able to chat with other buyers
- Different appearance of the map

Design specification

Login screen

Object	Details	Additional features	Location on screen
Title	Icon and “Buy locally” text	Green colour	Upper half
Two input boxes	Two boxes for email and password		Middle
Sign up button		Redirects to register page	Bottom right
Forgotten password button	Enables the user to change the password		Bottom left
Sign in button	Validates and checks the details given	Green colour, redirects to main page	Below text boxes

Register screen

Object	Details	Additional features	Location on screen
Text input boxes	Allow the user to put all the necessary information		Top and middle
Continue button	Writes the user’s details to the database, validates the details	Green colour	Middle
Already have an account button		Redirects to login page	Middle

Main page

Object	Details	Additional features	Location on screen
Top bar	It contains the title and on the left it has the button to bring the sidebar	Green colour	Top
Map	Map from Google with location of the user indicated		Middle
“Locate me”	A button that brings the user’s location to the middle of the screen	Ask for permission to use GPS	Bottom left of the map
Location pins	They are all across the map where the different producers are	Different colours indicate different product that are sold, when clicked redirect to the producer’s page	Middle
Bottom bar	It contains the legend to the map - what the colours of the pins mean	Green colour	Bottom

Main page sidebar

Object	Details	Additional features	Location on screen
“X” button	A button to close the sidebar and return to the main page	Green colour	Top right
User’s profile	A segment containing the user’s profile picture, name and email address		Top
Home button	A button to return to the home page	Green colour	Middle
Chats button	A button that redirects to the chats page	Green colour	Middle

Favourites button	A button that redirects to the favourites page	Green colour	Middle
Settings button	A button that redirects to the settings	Green colour	Bottom

Chats screen

Object	Details	Additional features	Location on screen
Top bar	It contains the title and on the left it has the button to go back to the main page, it also has searching button on the right to look for a certain chat	Green colour	Top
Chats	Each chat segment contains the producers' profile picture, name, last message and the time of the last message	When clicked redirects to the individual page of conversation	Middle and bottom

Individual chat screen

Object	Details	Additional features	Location on screen
Top bar	It contains the title and the name of the person we chat with	Green colour	Top
Messages	Each message segment contains of the message itself and the time it was sent		Middle
Box for user's message	A box where a user can input their message and a	When the sending button is clicked the message is sent,	Bottom

	button on the right to send the message	green colour	

Favourites screen

Object	Details	Additional features	Location on screen
Top bar	It contains the title and on the left it has the button to go back to the main page, it also has searching button on the right to look for a certain producer out of the favourite ones	Green colour	Top
Producers	Each producer's segment contains profile picture, name, what they sell, address	When clicked redirects to the individual page of the producer	Middle and bottom
Chat buttons	There are in each producer's segment, they redirect to the chat with the producer	When clicked redirects to the individual page of the producer	Bottom of each producer's segment

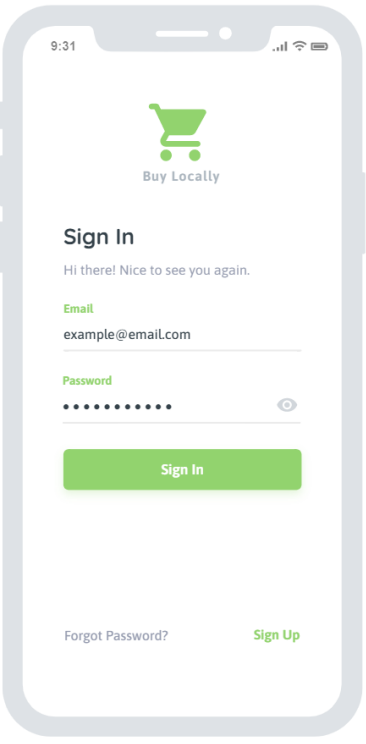
Producer's page screen

Object	Details	Additional features	Location on screen
Top bar	It contains the title and on the left it has the button to go back to the main page	Green colour	Top
Producer's details	Each producer's segment contains profile picture, name, what they sell, address		Middle and bottom
Producer's	It contains a		Right of the

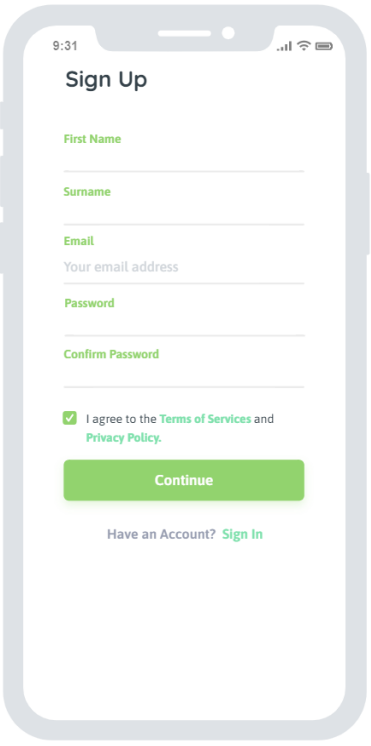
description	description of the given producer		producer's details
Review statistics	Contains the average review out of 5 stars, the number of reviews and visual representation with stars	Green colour of stars	Bottom of the producer's details
Chat button	There is a button to redirect to the chat with the producer	When clicked redirects to the individual page of the producer	Bottom right of the producer's details
Producer's products	A list of products a given producer is offering, each product segment contains picture, name, brief description and price		Middle
Give a review button	A button that allows the user to review the given producer	Green colour	Bottom

Final design

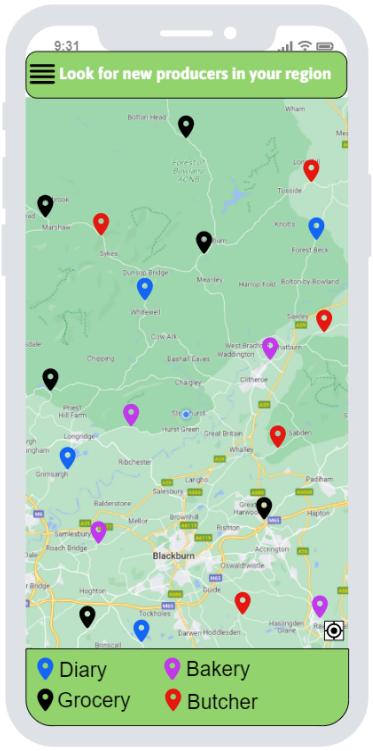
Login screen



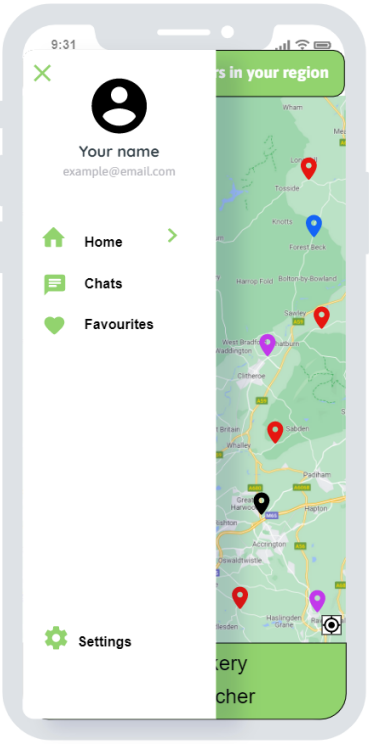
Register screen



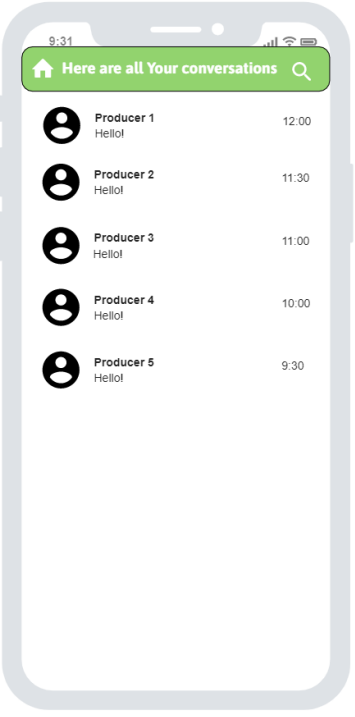
Main screen



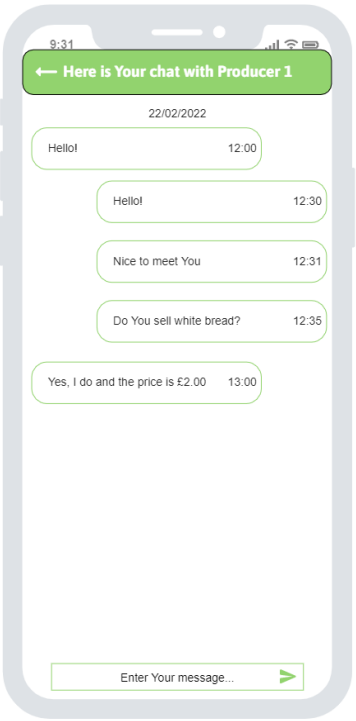
Main screen sidebar



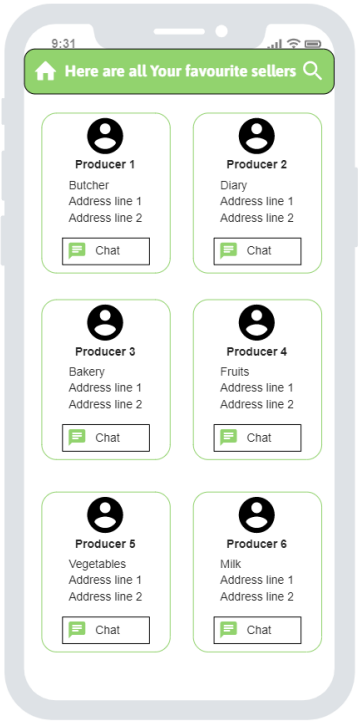
Chats screen



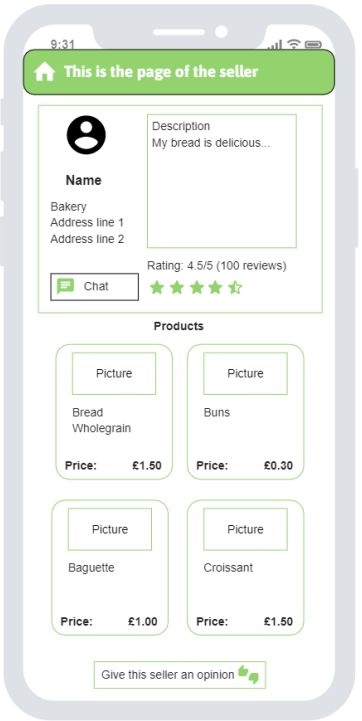
Individual chat



Favourites page



Producer's page



Interface feedback

I showed my final designs to the stakeholders – Daniel Uko and Szymon Lis to make sure they like them. I also asked for some aspects that should be changed.

Daniel Uko - the design seems very accessible and intuitive, and I really like the look of the map. It seems familiar and easy to use. I also think the main background colour should be calmer, not just straight-up white.

Szymon Lis - the layout of the design is fine, but I would choose a little less bright colour as it might ruin the experience a bit. Besides that, I think the design is clear and overall good.

I found the responses to be mostly positive. As for the colour changes, I might incorporate them. I will use a darker tone of green and more beige colour of the backgrounds in the chat section and producers' pages.

Validation

The program will validate most of the data entered by the user - listed below:

Validation algorithm	Justification
Validate of the password is secure enough - contains at least one capital letter, a number and length of eight characters	This validation will prevent the users from creating a password that is too simple, doesn't contain numbers, capital letters or is too short. A more difficult password means it will be more secure and more difficult to guess by unwanted people.
Validate that input boxes have not been left empty, in login and registration	This algorithm is needed as empty fields could break the database and should not be accepted as input. I cannot use empty strings as any of the details.
Validate if the enter message box contains something	This algorithm is needed to prevent sending empty messages in the chats section, so that if the message is empty and the user clicks the send button even accidentally, the empty message is not sent, which would be useless and obstruct the chat window.
Validate details when a user is trying to retrieve a password they have forgotten	When the user is changing their password, we have to check if the validation code, name, and surname are correct to make sure only the user with the access to the email account can change the password and no unwanted people can tamper with other accounts.

Algorithms

Validation algorithms

Pseudocode	Explanation
WHILE true IF length(password)>=8 IF lowercase(password)!=password IF password contains a digit OUTPUT	The first validation algorithm will be used during registration. It will enforce that the password entered by the user is secure enough, which decreases the risk

success_message
BREAK
 	
password_box.outline = red
OUTPUT error_message
 	
// Clearly indicate to the user that they have to create a password according to the security rules
 	
ENDWHILE
 	
// Only stops if the password follows all the rules
 	
END PROCEDURE
	of an account being stolen.

Pseudocode	Explanation

WHILE any of input boxes are null
Empty_boxes.outline = red
OUTPUT error_message
 	
// Clearly indicate to the user that they have to fill all the boxes
 	
ENDWHILE
 	
// Only stops if all boxes are filled
 	
END PROCEDURE
	The second validation algorithm is used to check if any of the boxes to input information is empty during registration. These fields cannot be empty or else the database might crash and the data will be incomplete.

Pseudocode	Explanation

WHILE message == null
message_box.outline = red
OUTPUT error_message
 	
// Clearly indicate to the user that the message cannot be empty
 	
ENDWHILE
	The third validation algorithm will be used in chat screen to check if the message the user tries to send contains some text because the user should not be able to send empty messages.

// Only stops if the message is not empty END PROCEDURE	

Pseudocode	Explanation

CONNECT to database
// Gain access to the database of accounts
 	
SELECT record WHERE username, first name and surname match the ones given by the user
IF number_of_rows > 0
OUTPUT success_message
// Username exists and the details are correct
 	
SELECT password from record
 	
RETURN password
 	
// Select and return the password
ELSE
OUTPUT error_message
 	
ENDIF
	The fourth and final validation algorithm will be used for the forgotten password process. The user will be able to recover their password if they give the correct email address, first name and surname.

System algorithms

I am going to prepare algorithms for all essential parts of the code of the application.

Here are the subproblems I am going to write the pseudocode of

- Check login details, by checking them against the user database
- Check if the username already exists in registration
- Displaying the producers which are currently on the part of the map
- Displaying favourite producers
- Displaying chats
- Displaying messages
- Displaying products

These algorithms form a solution by connecting them altogether in the program. First thing the users will see when they open the app is the login page, which will use the first

algorithm to validate the details. On the bottom of this page, there will be a button for registering an account, where the second algorithm will be used to disallow multiple accounts of the same email address. Validation algorithms will also be used to make sure the details given are correct. When the user successfully logs in, they can access the map, which is the essential part of the application. On the map, only the producers that are currently visible should be displayed to save resources. When they add producers to favourites, they should be able to see a list of favourite producers on the main page, of which the fourth algorithm will take care. When the user decides to go into the chats section, all the chats of that user will be displayed using the fifth algorithm. Each chat window will display the messages in the currently selected conversation using the sixth algorithm. And lastly, on the producer profile page, all their products will be listed using the seventh algorithm. Below are the pseudocodes of all the system algorithms.

Pseudocode	Explanation

CONNECT to database
// Gain access to the database of accounts
 	
SELECT record WHERE username=input.username
// Return the matching record
 	
IF record[password] == input.password
Success = true
OUTPUT success_message
// If the password matches the original one the user is allowed access
ELSE
Success = false
OUTPUT error_message
// If the password does not match or the user is not in the database the user is denied access
 	
ENDIF
 	
RETURN success
	This first system algorithm is used for logging the user. It checks if the username exists and if the password is correct - the same as the one given during registration. This way only authorised people can log in.

Pseudocode	Explanation
CONNECT to database // Gain access to the database of accounts WHILE true SELECT record WHERE	This second system algorithm will check if the username given by the user is already in use - we do not want two users with the same username. Username should be a

username=input.username
// Return the matching record
 	
IF record == null
OUTPUT success_message
BREAK
 	
username_box.outline = red
OUTPUT error_message
 	
// Clearly indicate to the user that they have to choose a different username that will be unique
 	
ENDWHILE
 	
// Only stops if the username chosen is unique
 	
END PROCEDURE
	unique key for every user.

Pseudocode	Explanation

CONNECT to database
// Gain access to the database of accounts
 	
SELECT users FROM TABLE Producers
// We only want to check the producers
 	
FOR EVERY producer IN users
// Iterate over all producers
IF on_screen(producer.location)
// Check if the producer should be displayed
display(producer)
// Display the user if needed
 	
END FOR
 	
END PROCEDURE
	The next system algorithm will be used to determine which producers are on the user screen to show their location

Pseudocode	Explanation
CONNECT to database	The next system algorithm will be

// Gain access to the database of accounts
 	
SELECT producers FROM TABLE Producers
// We only want to check the producers
 	
User = Current_user
// Set the user to the current user that is using the app
 	
FOR EVERY producer IN producers
// Iterate over all producers
IF producer IN User.favourites
// Check if the producer should be displayed
display(producer)
// Display the user if needed
 	
END FOR
 	
END PROCEDURE
	used to determine which users are the favourites of a particular buyer and display them in the favourites section

Pseudocode	Explanation

CONNECT to database
// Gain access to the database
 	
SELECT chats FROM TABLE Chats
// We only want to check the chats
 	
User = Current_user
// Set the user to the current user that is using the app
 	
FOR EVERY chat IN chats
// Iterate over all chats
IF chat.sender == User OR chat.recipient == User
// Check if the chat should be displayed
display(chat)
// Display the chat if needed
 	
END FOR
 	
END PROCEDURE
	The next system algorithm will be used to determine which chats are between the current user and a producer

Pseudocode	Explanation

CONNECT to database
// Gain access to the database
 	
Chat_ID = current_chat
// Set the chat to the current chat that the user is in
 	
SELECT messages FROM TABLE Chats.messages
WHERE ID=Chat_ID
// We only want to check the messages from the particular chat the user currently uses
 	
FOR EVERY message IN messages
// Iterate over all messages
IF place_on_screen(message)
// Check if the message should be displayed
display(message)
// Display the message if needed
 	
END FOR
 	
END PROCEDURE
	The next system algorithm will be used to determine which messages between the buyer and the producer should be displayed based on checking if there is place on the screen

Pseudocode	Explanation

CONNECT to database
// Gain access to the database
 	
Producer = current_producer
// Set the producer to the current producer the user is checking
 	
SELECT products FROM TABLE Products WHERE owner=Producer
// We only want to check the products that belong to the particular producer
 	
FOR EVERY product IN products
// Iterate over all products
IF place_on_screen(product)
// Check if the product should be displayed
display(product)
	The next system algorithm will be used to determine which messages products of the particular product, which the user is browsing, to display and also checking if there is the place on the screen to display it

// Display the product if needed
 	
END FOR
 	
END PROCEDURE	

Usability features

My application has a lot of user experience as almost every part of it is presented to the user. The design of the screens is very intuitive, as it takes at most 3 clicks to get to the desired destination. This links back to the usability features, as they can be further justified by the user experience needs.

Usability feature	Justification
The ability to search by producers, products, or postcode	Searching by producer name, product or postcode grants the users the necessary ease of finding producers in any way that is comfortable to them. This will ensure the producers find desired producers as quickly as possible.
The map, which is the main part of the application	The map is an instance of the Google Maps map, which the majority of users are familiar with, which makes the experience smoother. While using the map, the user is able to locate themselves with a single click. They can zoom in and out and move around, which is just what they would expect.
Meaningful icons on the map	The icons on the map have different colours indicating the type of product being sold, which makes choosing a producer from the map much easier, as the users know immediately what a given producer is selling. When they click an icon, a box pops up, which when clicked redirects to the page of the producer chosen.
Producer page with all details	On the page of the producer, their rating is displayed right next to the description and below all their products are listed. There is also a button to quickly give a review. This will ensure the users have easy access to the producer's description, products and the rating, as

	well as an option to express their feelings by leaving a review.
Login, register and forgotten password screens	Working and robust authentication pages are the foundation of user experience, because that is the first thing they see when opening the application. The login screen has to include buttons to allow for quick account creating or password retrieval.
Chats section	Chatting will be one of essential features as it will give the users the most convenient way to contact producers on the spot. Every chat should be listed with the avatar of the given producer, the last message in this chat and the time it was sent. When the user clicks the conversation, they are taken to the individual chat screen. Working chats list will ensure the users have an intuitive and pleasant experience, especially when they want to contact many different producers at once.
The ability to go back to the main page at any point	There should be an option to return to the main menu at any time to ensure simplicity and comfortability of use. This will make the application more user-friendly, because they can always return to the main page, where all their favourite producers are listed.

Data structures

Data structure	Explanation
Stack	To implement a feature of undoing the last action I will need a stack, which is suitable for this as a stack is a Last in First Out data structure. Every action will be pushed onto the stack and then if we want to backtrack, we just pop the last action from the stack.
Arrays	Arrays will be useful as they grant a way of quickly retrieving the information of a particular product, for example.
Tuples	Tuples might be useful to store the location of every producer because the location is

	static and not changeable during runtime.

Variables

LoginUser.java

Name	Data type	Explanation
goToRegister	TextView	Clicking this text redirects to the register page
forgotPassword	TextView	Clicking this text redirects to the forgotten password page
goToProducer	TextView	Clicking this text redirects to the producer register page
email	EditText	This is the input box for user's email address
password	EditText	This is the input box for user's password
login	Button	Clicking this button initializes the attempt to sign the user in
str_email	String	The String value of what the user has inputted into the email field
str_password	String	The String value of what the user has inputted into the password field
databaseHelper	DBHelper	An instance of the class created by me to be able to access and write to the database
allMatching	List<CustomerModel>	The returned list of customers whose details match the inputted details
found	CustomerModel	The first customer in the list above, assigned if the list is not empty, assigned to allMatching.get(0)

RegisterUser.java

Name	Data type	Explanation
btn_register	Button	Clicking this button registers a new user if the details are correct
btn_gotoLogin	Button	Clicking this button redirects to the login page
et_firstName	EditText	Input box for user's first name
et_surname	EditText	Input box for user's surname
et_email	EditText	Input box for user's email address
et_password	EditText	Input box for user's password
et_confirmPassword	EditText	Input box for user's confirmed password
str_firstName	String	The String value of what the user has inputted into the first name field
str_surname	String	The String value of what the user has inputted into the surname field
str_email	String	The String value of what the user has inputted into the email field
str_password	String	The String value of what the user has inputted into the password field
str_confirmPassword	String	The String value of what the user has inputted into the confirm password field
databaseHelper	DBHelper	An instance of the class created by me to be able to access and write to the database
customer	CustomerModel	A new instance of the customer class for a new customer added to the system
success	Boolean	The outcome of the function that adds the new customer to the database

MainActivity.java

Name	Data type	Explanation
goToMap	Button	Clicking this button redirects to the map
accountEmail	TextView	Text to display the current user's email address
fullName	TextView	Text to display the current user's full name
favouritesList	RecycleView	Android's version of visual lists that can be scrolled and items added to
menu	Menu	Android's menu (three dots in the top right) for sorting the list of favourite producers as needed
allFavourites	List<ProducerModel>	The actual list of favourite producers of the current user that is transferred to visuals in the RecycleView
sharedPreferences	SharedPreferences	An instance of SharePreferences that allows to retrieve the email address of the current user
email	String	Using SharedPreferences, the email of the current user is retrieved
databaseHelper	DBHelper	An instance of the class created by me to be able to access and write to the database
found	CustomerModel	The first customer with the current user's email address found in the database
allFavouritesIDs	List<Integer>	Using the databaseHelper we can get the IDs of all favourite producers of the current user

UML diagrams

Classes

User	Producer inherits User	Buyer inherits User
+ name : string + surname : string + email : string + hashed_password : string + profile_picture : image	+ location : pair<string, string> + conversations : class:conversation + description : string + rating : integer + no_reviews : integer	+ favourite_producers : list<Producer> + location : pair<string, string> + conversations : class:conversation
+ set_name() + set_surname() + get_name() + get_name() + set_email() + get_email() + set_password() + get_password()	+ update_rating() + show_location() + initialize_producer() + get_conversations()	+ add_favourite() + initialize_buyer() + set_location() + get_conversations()

Product
+ description : string + price : float + picture : image + seller : Producer
+ add_description() + display_price() + initialize_product() + display_picture()

Conversation
+ producer : Producer + buyer : Buyer + last_message : string + last_date : date + avatar : image + messages : list<message>
+ print_message() + display_avatar() + initialize_cenversation()

Message
+ sender : User + recipient : User + content : string + time : time + picture : image
+ send_message() + print_content() + show_time() + send_picture() + initialize_message()

Test strategy

Iterative testing

As I will be carrying out iterative development, meaning I will take a section of the project, develop it, test it and repeat this process until the finished prototype is made. After the development of each prototype, I will complete testing of the new features incorporated using the test data.

The type of testing during iteration I will be using is white box testing. This is dependent on the code logic and tests each possible path at least once. This is also alpha testing, as it will be carried out by me – the developer.

Test data

Test number	Process tested	Input	Type of input	Expected output	Justification
1	Splash screen	Opening the app	N/A	Splash screen animation playing for 3 seconds then redirecting to login	The splash screen is before everything else in the app
2	Logging in	Correct email and password	Normal	Success, log in the user	I have to make sure the details given are correct
3	Logging in	Incorrect email address	Erroneous	Error, does not log in	To not give access user with incorrect details

4	Logging in	Incorrect password	Erroneous	Error, does not log in	To not give access user with incorrect details
5	Logging in	No input	Erroneous	Error, no details were given	To make sure the user is told to input the details when they try to leave them blank
6	Forgotten password	Correct details	Normal	Success, store the new password	To make sure the user is able to change the password if the details are correct
7	Forgotten password	Incorrect details	Erroneous	Error, no change in password	To make sure the user cannot change the password whn the details are incorrect
8	Registering	Password match	Normal	Success, add the new user to database	To make the account can be created when the passwords match
9	Registering	Passwords do not match	Erroneous	Error, passwords have to be the same	To make sure the user cannot register when the passwords do not match
10	Registering	No input	Erroneous	Error, no details were given	To make the user cannot leave the registration boxes blank
11	Registering	Wrong email	Erroneous	Error, display message	There cannot be two accounts on the same email address
11	Sidebar button	Click of button	N/A	Bring the sidebar to the screen	The user has to be able to move between sections
12	Chats button	Click of button	N/A	Take the user to chats section	The user has to be able to move to chats section
13	Favourites button	Click of button	N/A	Take the user to favourites page	The user has to be able to move to favourites section
14	Sending messages	Text message	Normal	Success, send the message to the	The make sure the user can send messages

				recipient	
15	Sending messages	No input	Erroneous	Error, no message inputted	To make sure the user cannot send empty messages

Post-development testing

In my post-development testing, I will use beta testing to test my system for intuitiveness, robustness, and usability. This will also be a form of black-box testing, as the end-users do not know anything about the code of the application. This will help me make sure all the features are as accessible and easy to use as possible and help me correct them. Additionally, this form of testing will allow me to see how the solution copes with real users who may do something unexpected and reveal some errors which I did not notice myself in the code. To test the agility, I will give the program to my 2 end users – Szymon and Daniel, who I will ask to try to break the program by ignoring instructions and entering incorrect data. To test for usability, I will ask the end-users the following questionnaire. This will allow me to test how easily my end-user could use the program and how effective they think the solution is.

Questionnaire

Question	Justification
Did you have trouble navigating the app's interface or finding the information you were looking for?	The user experience is crucial to the success of my app. By asking this question, I can identify any areas where the app's interface or organisation may need improvement. If users have difficulty navigating the app or finding the information they require, they may become frustrated and give up on the app. By addressing these issues, I can improve the overall user experience and increase engagement with my app.
Did you encounter any errors or bugs while using the app?	Technical issues can be a major source of frustration for users. By asking this question, I can identify any errors or bugs that users may encounter while using the app. I can use this information to improve the app's stability and ensure that users have a seamless experience while using the app.

Was it easy to create an account and log in to the app?	The account creation and authentication process is a critical aspect of the app's usability. The authentication process has to be as straightforward as possible. By asking this question, I can identify any issues with the account creation and authentication process and make improvements to ensure that users can easily create an account and log in to the app.
Are you able to add the producers to your favourites and see the list on the main page?	This question is important to test the ease and effectiveness of the app's favourites feature. The users should not have any difficulty adding producers to their favourites or finding their list of favourites on the main page. This feature can help users keep track of producers they are interested in and make it easier for them to find and purchase products from those producers in the future.
Was it easy to view and navigate the map, and were the producer locations clearly marked and easy to find?	The usability of the app's map feature is critical to the success of the app. If users have difficulty viewing or navigating the map, or if the producer locations are not clearly marked or easy to find, they may become frustrated. By asking this question, I can identify any issues with the app's map feature and make improvements to ensure that users can easily view and navigate the map.
Do you like the feel of the application? What could be improved or more intuitive?	This last question is a broader one, here the end-users can express their feelings with the use of my application and comment on any features they don’t like or which should be added or made more intuitive.

I will send the questionnaire below to both Szymon and Daniel and ask them to fill it in to gather their feedback.

Did you have trouble navigating the app's interface or finding the information you were looking for?

Did you encounter any errors or bugs while using the app?

Was it easy to create an account and log in to the app?

Are you able to add the producers to your favourites and see the list on the main page?

Was it easy to view and navigate the map, and were the producer locations clearly marked and easy to find?

Do you like the feel of the application? What could be improved or more intuitive?

Development

Breaking down the project into iterative model

I will be programming my project in prototypes, which will include the following features in order: CustomerModel class, splash screen, DatabaseHelper class, Login Activity, Register Activity, Forgot Password Activity, Basic Map Activity, ProducerModel class, Producer Register Activity, drawing producers' markers on the map, Producer Profile Activity, adding producers to favourites, Home Page Activity with sorting. The choice of order is connected to the decomposition diagram I created during the design stage, and also by the flow of the development, which requires me to create the main classes - CustomerModel, ProducerModel and DatabaseHelper at the beginning together with the splash screen which is the first thing the user sees when they open the application. Also, I will leave the favourite producers feature for the end, as it requires all the previous features to be already implemented and tested.

First Prototype

I started with a class for the customer class. I created two constructors in case I need a different one along the way. A customer has its ID, which is unique, first name, surname, email address and password to the account. I also created the toString method, which enables me to print user details in a much nicer way. I am creating the customer class because it will handling the users much easier as all the details will be in one entity.

```java
package com.example.firstprototype;

public class CustomerModel {

    private int id;
    private String firstName;
    private String surname;
    private String email;
    private String password;

    // constructors

    public CustomerModel(int id, String firstName, String surname, String email, String password) {
        this.id = id;
        this.firstName = firstName;
        this.surname = surname;
        this.email = email;
        this.password = password;
    }

    public CustomerModel() {
    }

    // toString for outputting the data of a customer

    @Override
    public String toString() {
        return "CustomerModel{" +
                "id=" + id +
                ", firstName='" + firstName + '\'' +
                ", surname='" + surname + '\'' +
                ", email='" + email + '\'' +
                ", password='" + password + '\'' +
                '}';
    }
```

Then I created all the getters and setters to be able to get values of attributes and to set values as attributes. Although I will probably not use all of them, it is useful if I require them later I will not have to create new ones. I will use the getters and setters to access the class variable.

```java
    // getters and setters

    public int getId() { return id; }

    public void setId(int id) { this.id = id; }

    public String getFirstName() { return firstName; }

    public void setFirstName(String firstName) { this.firstName = firstName; }

    public String getSurname() { return surname; }

    public void setSurname(String surname) { this.surname = surname; }

    public String getEmail() { return email; }

    public void setEmail(String email) { this.email = email; }

    public String getPassword() { return password; }

    public void setPassword(String password) { this.password = password; }
}
```

Then I started creating the splash screen of the app, with a gif image and the name of the app. This will make opening the app coller and more interesting. The design is as follows:

Then I created the Java code of the splash screen activity. It is supposed to play the gif image of a green shopping cart for 3 seconds and then redirects the user to the login activity where the user can also choose to sign up.

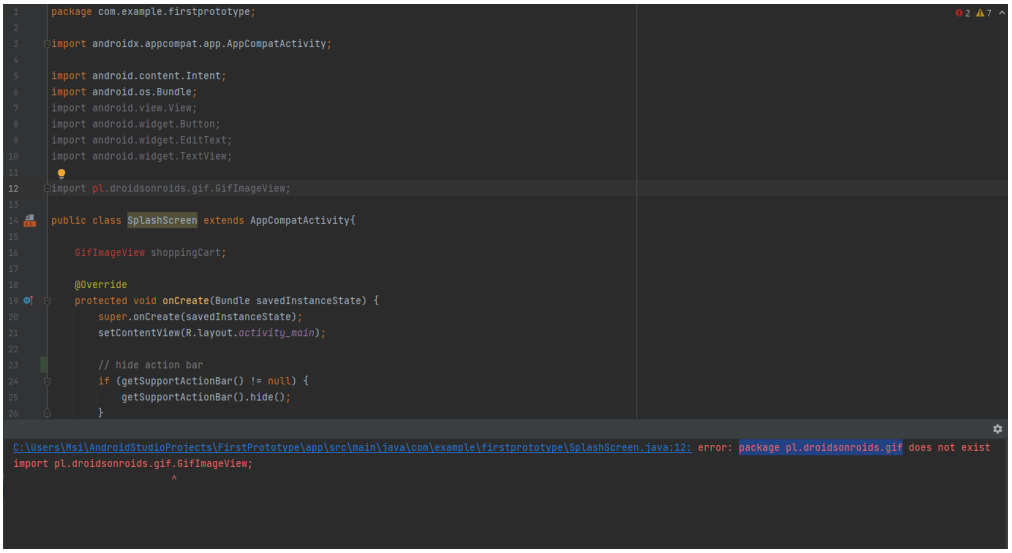
```java
package com.example.firstprototype;

import ...

public class SplashScreen extends AppCompatActivity{

    GifImageView shoppingCart;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // hide action bar
        if (getSupportActionBar() != null) {
            getSupportActionBar().hide();
        }

        // create an Intent to send the user to after 3 seconds
        final Intent i = new Intent( packageContext: SplashScreen.this, LoginUser.class);

        // timer for waiting 3 second while the gif file plays
        Thread timer = run() -> {
                try{
                    sleep( millis: 3000);
                } catch (InterruptedException e) {
                    e.printStackTrace();
                } finally {
                    startActivity(i);
                    // we do not need the splash screen activity anymore
                    finish();
                }
        };

        // start the timer
        timer.start();

    }
}
```

To display a GIF I use a package - package pl.droidsonroids.gif. Then I ran the app to test the splash screen activity, to check if the GIF file plays for 3 seconds, and then it is automatically redirected to the login page to set a fixed amount of time until redirecting to the main screen every time.

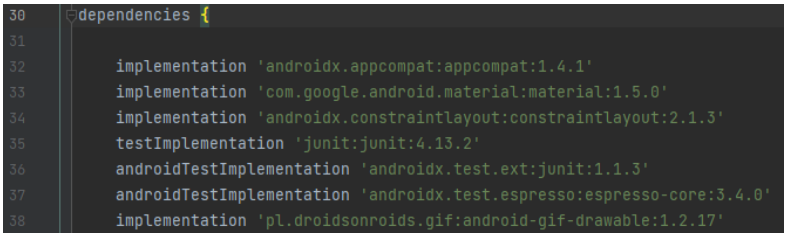
Testing table

Test number	Process tested	Input	Type of input	Expected output	Actual output

1	Splash screen	Opening the app	N/A	Splash screen animation playing for 3 seconds then redirecting to login	The app does not run, there is an error



The app cannot contain an issue which prevents the users from accessing the program, and so I have to fix this error. To resolve the issue, I had to import the package to use it. So I did just that - I added a new dependency to the build.gradle file. The last line on this screen is the newly added implementation of a library handling GIFs.



I ran the app and got no errors. The GIF plays for 3 seconds and the activity redirects to the login user activity. Before creating the login and register activities, I created the database helper class. It creates the SQLite database and uses SQL statements to add records to it and find them.

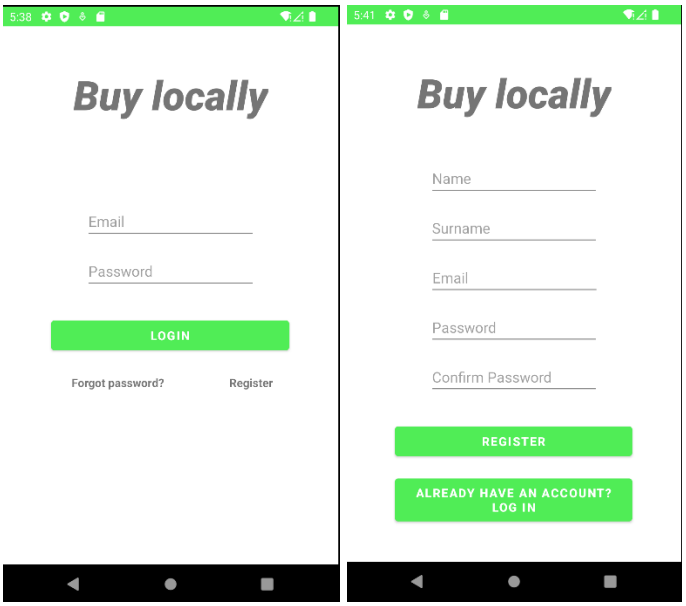
```java
package com.example.firstprototype;

import ...


// this class implements the SQLite database

public class DBHelper extends SQLiteOpenHelper {


    public static final String CUSTOMER_TABLE = "CUSTOMER_TABLE";
    public static final String COLUMN_FIRSTNAME = "FIRSTNAME";
    public static final String COLUMN_SURNAME = "SURNAME";
    public static final String COLUMN_EMAIL = "EMAIL";
    public static final String COLUMN_PASSWORD = "PASSWORD";
    public static final String COLUMN_ID = "ID";

    // constructor of the class
    public DBHelper(@Nullable Context context) { super(context, name: "customers.db", factory: null, version: 1); }


    // this is called the first time database is accessed
    @Override
    public void onCreate(SQLiteDatabase db) {

        // create the SQL statement of creating a table
        String createTableStatement = "CREATE TABLE " + CUSTOMER_TABLE + " (" + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " + COLUMN_FIRSTNAME + " TEXT, " + COLUMN_SURNAME + " TEXT, " + COLUMN_EMAIL + " TEXT,

        // execute the statement
        db.execSQL(createTableStatement);

    }

    // this is called whenever the version number of the database changes
    @Override
    public void onUpgrade(SQLiteDatabase db, int i, int i1) {

        db.execSQL("DROP TABLE " + CUSTOMER_TABLE);
        onCreate(db);

    }
```

```java

    // registration
    public boolean addCustomer(CustomerModel customer){

        // get the access to the database
        SQLiteDatabase db = this.getWritableDatabase();
        ContentValues cv = new ContentValues();

        // insert the data to content values
        cv.put(COLUMN_FIRSTNAME, customer.getFirstName());
        cv.put(COLUMN_SURNAME, customer.getSurname());
        cv.put(COLUMN_EMAIL, customer.getEmail());
        cv.put(COLUMN_PASSWORD, customer.getPassword());


        // add the record to the database and get the result
        long insert = db.insert(CUSTOMER_TABLE, nullColumnHack: null, cv);
        db.close();

        // return the correct boolean result
        if(insert == -1){
            return false;
        }
        return true;
    }


    // get a list of all accounts with given email
    public List<CustomerModel> getAllMatching(String checkedEmail){

        List<CustomerModel> retList = new ArrayList<>();

        // get the data from the database
        String selectStatement = "SELECT * FROM " + CUSTOMER_TABLE + " WHERE " + COLUMN_EMAIL + " = '" + checkedEmail + "'";

        // get access to the database
        SQLiteDatabase db = this.getReadableDatabase();

        // do a query to search the database
        Cursor cursor = db.rawQuery(selectStatement, selectionArgs: null);
```

```java

        if(cursor.moveToFirst()){
            // loop through the result and create new customers and add them to the list
            do {
                // convert all cursor values to their types
                int customerID = cursor.getInt( i: 0);
                String customerFirstName = cursor.getString( i: 1);
                String customerSurname = cursor.getString( i: 2);
                String customerEmail = cursor.getString( i: 3);
                String customerPassword = cursor.getString( i: 4);

                // create the customer object and add it to the list

                CustomerModel newCustomer = new CustomerModel(customerID, customerFirstName, customerSurname, customerEmail, customerPassword);
                retList.add(newCustomer);

            } while (cursor.moveToNext());

        } else {
            // the list will just be empty - no matching results
        }

        // close both the database and the cursor
        cursor.close();
        db.close();
        return retList;
    }


}
```

Then I created the login and register activity design. There is the name of my app, all input fields and buttons accordingly. This is the first main screen the users will see when opening the app, so they have to look tidy.

Login activity and register activity designs:

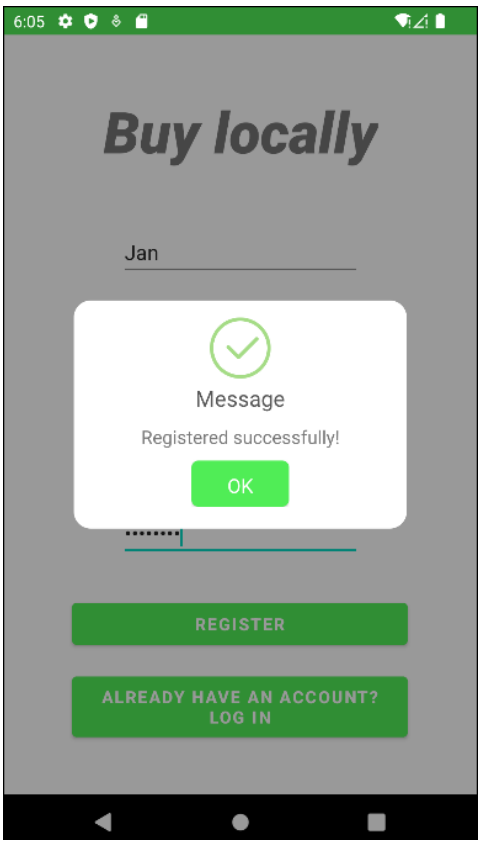


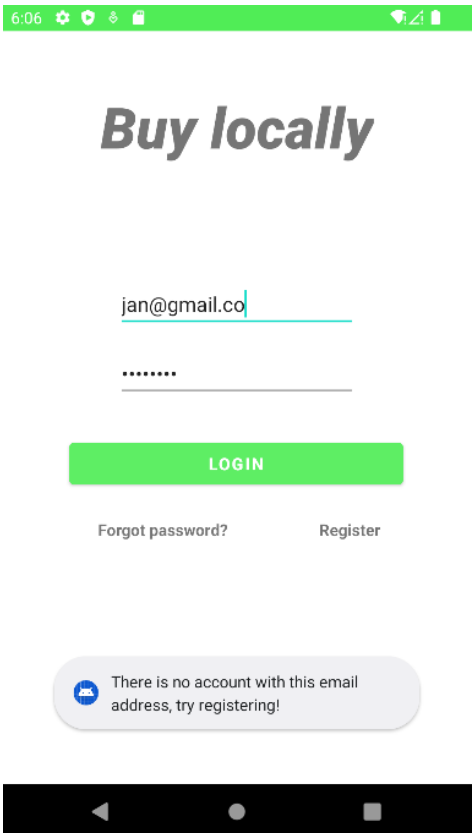
Login activity code:

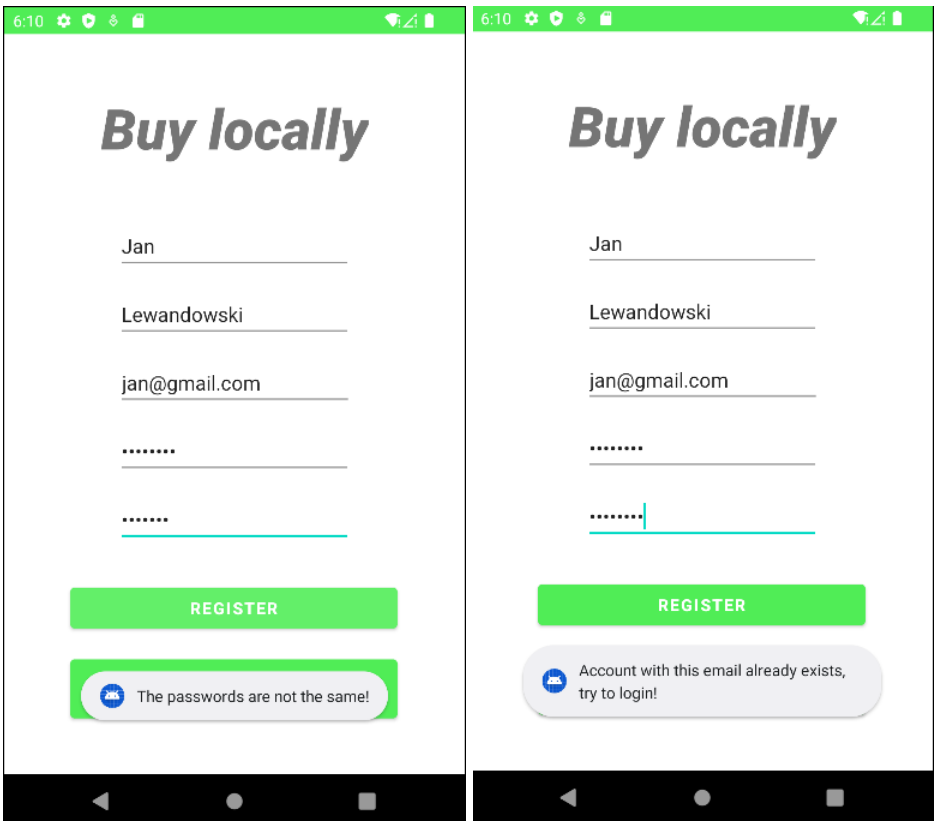
```java
package com.example.firstprototype;

import ...

public class LoginUser extends AppCompatActivity {

    // references to all buttons and other controls on the layout
    TextView goToRegister, forgotPassword;
    EditText email, password;
    Button login;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login_user);

        // hide the action bar
        if (getSupportActionBar() != null) {
            getSupportActionBar().hide();
        }

        // assign every element on the screen
        goToRegister = findViewById(R.id.goto_register);
        forgotPassword = findViewById(R.id.forgotPassword);

        email = findViewById(R.id.email);
        password = findViewById(R.id.password);

        login = findViewById(R.id.login);

        // set on click listeners

        login.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) { loginUser(); }
        });

```

```
    goToRegister.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            final Intent i = new Intent( packageContext: LoginUser.this, RegisterUser.class);
            startActivity(i);

        }
    });
}

private void loginUser() {
    // convert all input into strings
    String str_email = email.getText().toString();
    String str_password = password.getText().toString();

    if(str_email.length() == 0 || str_password.length() == 0){
        Toast.makeText( context: LoginUser.this, text: "Enter email and password!", Toast.LENGTH_SHORT).show();
    } else {
        // get the database
        DBHelper databaseHelper = new DBHelper( context: LoginUser.this);

        // find all matching records
        List<CustomerModel> allMatching = databaseHelper.getAllMatching(str_email);

        // check if an account of this email already exists
        if (allMatching.size() == 0) {
            Toast.makeText( context: LoginUser.this, text: "There is no account with this email address, try registering!", Toast.LENGTH_SHORT).show();
        } else {
            CustomerModel found = allMatching.get(1);


            if (!found.getPassword().equals(str_password)) {
                Toast.makeText( context: LoginUser.this, text: "Incorrect password!", Toast.LENGTH_SHORT).show();
            } else {
                Intent i = new Intent( packageContext: LoginUser.this, HomeActivity.class);
                startActivity(i);
            }
        }
    }

}
```

App is running correctly, I am able to register an account and log in, which is one of the essential features. I registered my account with the email address jan@gmail.com and password 1234567A. When I try to log in with an incorrect email - jan@gmail.co, or two passwords are not the same, a correct error message is displayed. Below is the app running:





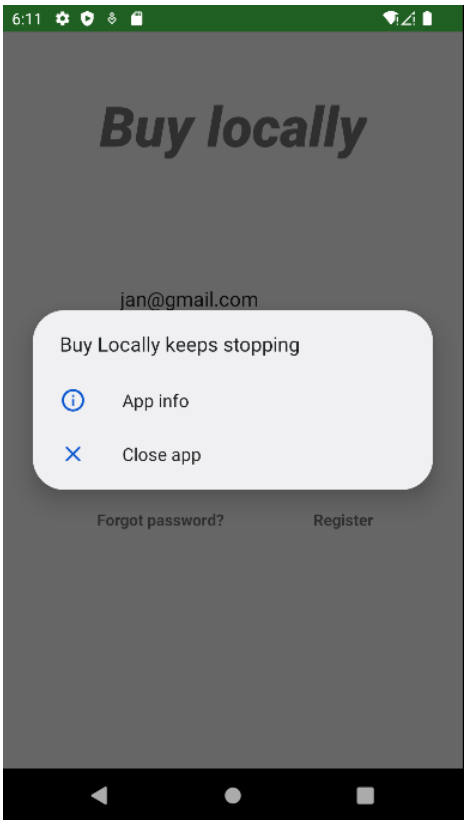


Testing table

Test number	Process tested	Input	Type of input	Expected output	Actual output
1	Splash screen	Opening the app	N/A	Splash screen animation playing for 3 seconds then redirecting to login	Correct
2	Logging in	Correct email and password	Normal	Success, log in the user	Error
3	Logging in	Incorrect email address	Erroneous	Error, does not log in	Success
4	Logging in	Incorrect password	Erroneous	Error, does not log in	Success
5	Logging in	No input	Erroneous	Error, no details were given	Success
6	Forgotten password	Correct details	Normal	Success, store the new	Not yet implemented

				password	
7	Forgotten password	Incorrect details	Erroneous	Error, no change in password	Not yet implemented
8	Registering	Password match	Normal	Success, add the new user to database	Success
9	Registering	Passwords do not match	Erroneous	Error, passwords have to be the same	Success
10	Registering	No input	Erroneous	Error, no details were given	Success
11	Registering	Wrong email	Erroneous	Error, display message account already exists	Success

I encountered an error while in the Login Screen I input the correct email and password of a previously created account. When I try to do that, the app crashes. This error would prevent users from logging in their accounts, so it must be fixed.



I remedied this error by first looking at the error message.

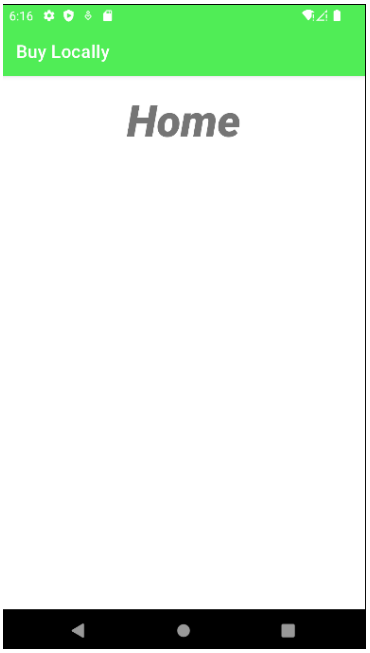
```
2022-05-02 19:11:18.354 27342-27342/com.example.firstprototype E/AndroidRuntime: FATAL EXCEPTION: main
    Process: com.example.firstprototype, PID: 27342
    java.lang.IndexOutOfBoundsException: Index: 1, Size: 1
        at java.util.ArrayList.get(ArrayList.java:437)
        at com.example.firstprototype.LoginUser.loginUser(LoginUser.java:80)
        at com.example.firstprototype.LoginUser.access$000(LoginUser.java:16)
        at com.example.firstprototype.LoginUser$1.onClick(LoginUser.java:48)
        at android.view.View.performClick(View.java:7441)
        at com.google.android.material.button.MaterialButton.performClick(MaterialButton.java:1131)
        at android.view.View.performClickInternal(View.java:7418)
        at android.view.View.access$3700(View.java:835)
        at android.view.View$PerformClick.run(View.java:28676)
        at android.os.Handler.handleCallback(Handler.java:938)
        at android.os.Handler.dispatchMessage(Handler.java:99)
        at android.os.Looper.loopOnce(Looper.java:201)
        at android.os.Looper.loop(Looper.java:288)
        at android.app.ActivityThread.main(ActivityThread.java:7839) <1 internal line>
        at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
        at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:1003)
```

I found the error – I was getting index 1 of the list, but the list only has 1 element, so I have to get index 0.

Quick fix and everything works now. I changed the index I was getting from 1 to 0 - the beginning of the list. When it was set to 1 it did not work because if the list had length 1, index 1 was out of the list which caused the error.

```
61
62      private void loginUser() {
63          // convert all input into strings
64          String str_email = email.getText().toString();
65          String str_password = password.getText().toString();
66
67          if(str_email.length() == 0 || str_password.length() == 0){
68              Toast.makeText( context: LoginUser.this,  text: "Enter email and password!", Toast.LENGTH_SHORT).show();
69          } else {
70              // get the database
71              DBHelper databaseHelper = new DBHelper( context: LoginUser.this);
72
73              // find all matching records
74              List<CustomerModel> allMatching = databaseHelper.getAllMatching(str_email);
75
76              // check if an account of this email already exists
77              if (allMatching.size() == 0) {
78                  Toast.makeText( context: LoginUser.this,  text: "There is no account with this email address, try registering!", Toast.LENGTH_SHORT).show();
79              } else {
80                  CustomerModel found = allMatching.get(0);
81
82
83                  if (!found.getPassword().equals(str_password)) {
84                      Toast.makeText( context: LoginUser.this,  text: "Incorrect password!", Toast.LENGTH_SHORT).show();
85                  } else {
86                      Intent i = new Intent( packageContext: LoginUser.this, HomeActivity.class);
87                      startActivity(i);
88                  }
89              }
90          }
91
92      }
93  }
```

When I log in with the correct details, I get redirected to the home page, which is empty for now. This means the error has been successfully resolved.

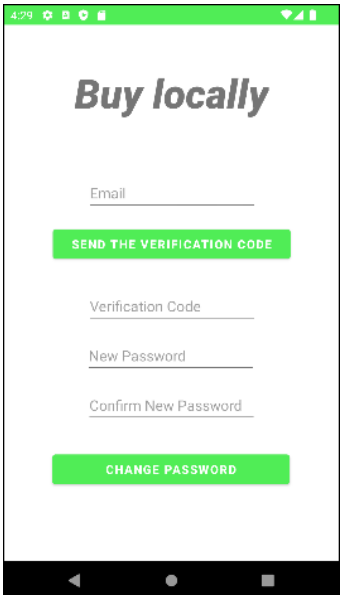


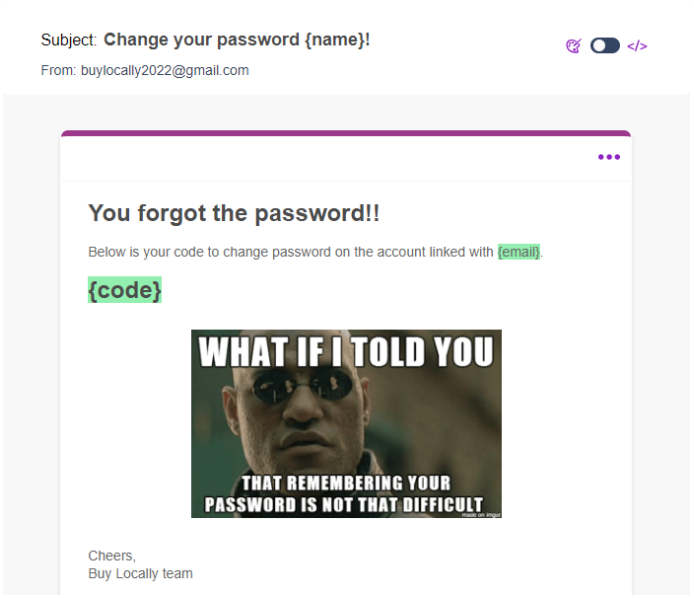
Second Prototype

In this second prototype, I focused on implementing the forgot password feature, which consists of sending emails with verification code to the user, and then updating the record in the database. The user receives the email, puts the verification code in the app, then they can input a new password and confirm it. After submitting the form, their password is changed in the database, and they can log in with their new password.

I started by creating the forgot password activity and designing its layout.

To send the emails to customers, I will be using Courier, where I designed an email template. The template includes spaces where I will be able to put data, such as the customer's name, email address and the verification code.





Then I created emails helper class. This class mainly consists of the code of the Courier Java API implementation, which can be found here. I need this code to be able to call the Courier API and send the email retrieval emails from my email account.

```java
public class EmailsHelper {

    public static void send(String Name, String Email, int Code) throws IOException {

        // create the thread for sending the email request to Courier
        Thread thread = new Thread(new Runnable() {

            @Override
            public void run() {
                try  {
                    // here I prepare the Courier request
                    Courier.init(ConfigEmail.CourierKey);

                    SendEnhancedRequestBody request = new SendEnhancedRequestBody();
                    SendRequestMessage message = new SendRequestMessage();

                    HashMap<String, String> to = new HashMap<~>();
                    to.put("email", Email);
                    message.setTo(to);
                    message.setTemplate("B4QNN7GN4P47C8MB4B19VJ8J1YF3");

                    // putting the data in
                    HashMap<String, Object> data = new HashMap<~>();
                    data.put("variables", "awesomeness");
                    data.put("email", Email);
                    data.put("name", Name);
                    data.put("code", Code);
                    message.setData(data);

                    // send the request
                    request.setMessage(message);
                    try {
                        SendEnhancedResponseBody response = new SendService().sendEnhancedMessage(request);
                        System.out.println(response);
```



```java
                        SendEnhancedResponseBody response = new SendService().sendEnhancedMessage(request);
                        System.out.println(response);
                    } catch (IOException e) {
                        e.printStackTrace();
                    }

                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });

        // run the thread to send the request
        thread.start();
    }
}
```

The next step was to code the forgot password class, which consists of two vital functions, sendCode(), and checkDetails().

```java
public class ForgotPassword extends AppCompatActivity {

    // declare all elements of the screen
    EditText email, verificationCode, newPassword, confirmPassword;
    Button sendCode, changePassword;

    private int Code = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_forgot_password);

        // hide the action bar
        if (getSupportActionBar() != null) {
            getSupportActionBar().hide();
        }

        // find all elements of the screen, inputs and buttons
        email = findViewById(R.id.email);
        verificationCode = findViewById(R.id.VerificationCode);
        newPassword = findViewById(R.id.password);
        confirmPassword = findViewById(R.id.confirmPassword);

        sendCode = findViewById(R.id.sendCode);
        changePassword = findViewById(R.id.changePassword);

        // set up click listeners to the buttons
        sendCode.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) { sendCode(); }
        });

        changePassword.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) { checkDetails(); }
        });
    }
```

sendCode() function:

```java
    private void sendCode(){
        String str_email = email.getText().toString();

        // initialise the database helper and find matching accounts
        DBHelper databaseHelper = new DBHelper( context: ForgotPassword.this);
        List<CustomerModel> allMatching = databaseHelper.getAllMatching(str_email);

        // no matching accounts
        if(allMatching.size() == 0){
            Toast.makeText( context: ForgotPassword.this,  text: "Account with this email address does not exist, try regis
            return;
        }
        // generate the verification code and convert it to string
        Code = new Random().nextInt( bound: 900000) + 100000;
        String str_code = Integer.toString(Code);

        // get the name and then use EmailsHelper class to send the message
        CustomerModel found = allMatching.get(0);
        String name = found.getFirstName();

        try {
            EmailsHelper.send(name, str_email, Code);
            Toast.makeText( context: ForgotPassword.this,  text: "Verification message sent!", Toast.LENGTH_SHORT).show();
        } catch (IOException e) {
            e.printStackTrace();
        }
```

And the checkDetails() function:

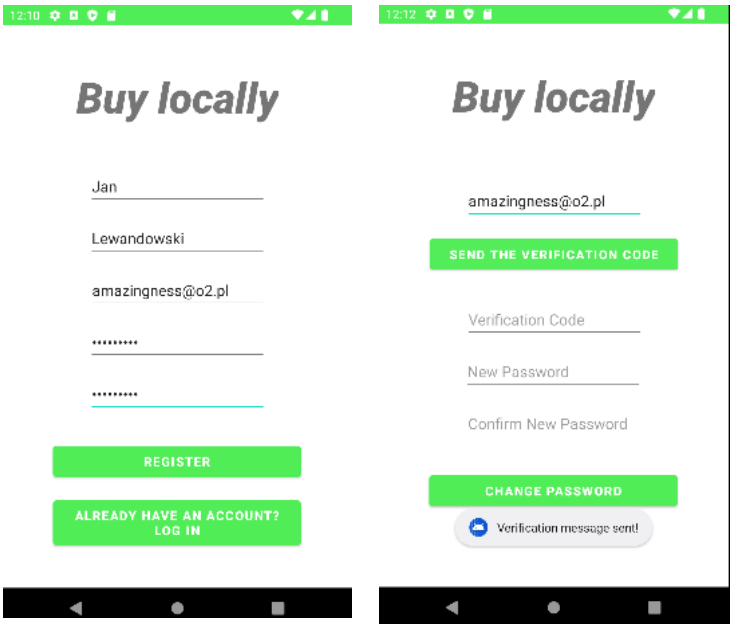
```
private void checkDetails(){
    // get the string values of all parameters
    String str_email = email.getText().toString();
    String str_verificationCode = verificationCode.getText().toString();
    String str_newPassword = newPassword.getText().toString();
    String str_confirmPassword = confirmPassword.getText().toString();
    String str_code = Integer.toString(Code);

    // verify the inputs
    if(!str_verificationCode.equals(str_code)){
        Toast.makeText( context: ForgotPassword.this,  text: "The verification code is not correct, try again!", Toast.
    } else if(!isPasswordValid(str_newPassword)){
        // check if password is secure enough
    } else if(!str_newPassword.equals(str_confirmPassword)){
        Toast.makeText( context: ForgotPassword.this,  text: "Passwords are not the same!", Toast.LENGTH_SHORT).show();
    } else {
        // initialize the database helper
        DBHelper databaseHelper = new DBHelper( context: ForgotPassword.this);

        // get all matching accounts and then the first of them
        List<CustomerModel> allMatching = databaseHelper.getAllMatching(str_email);
        CustomerModel found = allMatching.get(0);

        // call the change password function with the new password and the customer ID
        boolean success = databaseHelper.changePassword(str_newPassword, found.getId());

        // using an alert to inform the user the password has been changed and move to login page
        new SweetAlertDialog( context: ForgotPassword.this, SweetAlertDialog.SUCCESS_TYPE)
                .setTitleText("Message")
                .setContentText("Password changed successfully!")
                .setConfirmText("OK")
                .setConfirmClickListener(new SweetAlertDialog.OnSweetClickListener() {
                    @Override
                    public void onClick(SweetAlertDialog sweetAlertDialog) {
                        final Intent i = new Intent( packageContext: ForgotPassword.this, LoginUser.class);
                        startActivity(i);
                    }
                })
                .show();
```

In the checkDetails() function, I need to use the database helper, to change a record in the database. I am passing the new password for the given user, and the ID, which I find by finding all accounts with the email address. Then I created the changePassword() function in the DBHelper class.

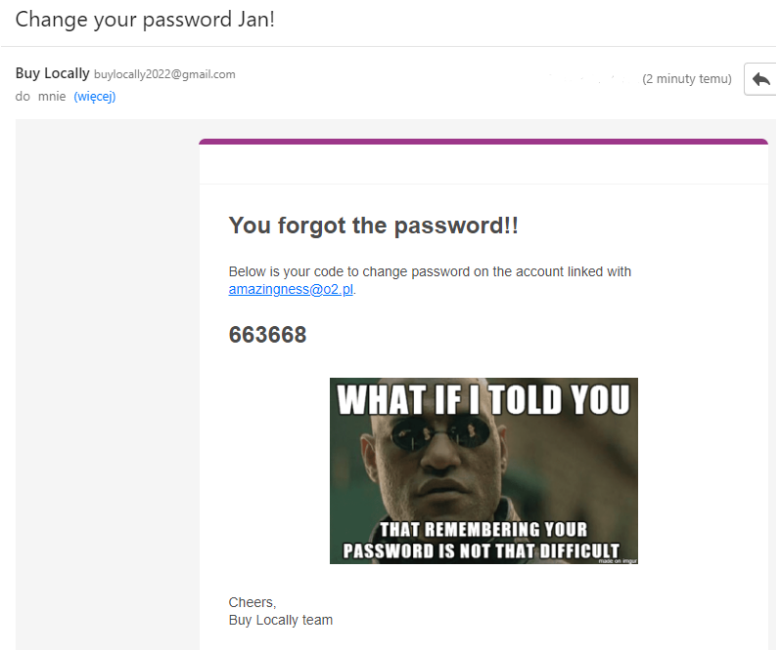
```
// changing password when forgotten
public boolean changePassword(String newPassword, int id){

    // get the access to the database
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv = new ContentValues();

    // insert the data to content values
    cv.put(COLUMN_PASSWORD, newPassword);

    // update the correct record
    long insert = db.update(CUSTOMER_TABLE, cv,  whereClause: COLUMN_ID + "=" + id,  whereArgs: null);
    db.close();

    // return the correct boolean result
    if(insert == -1){
        return false;
    }
    return true;

}
```

This function is responsible for changing the password in a record where the ID matches. To do that, I first had to get the database in writable form and create content values, where I put the details that needed to be changed. Then I execute the SQL statement using the db.update function. After that, I close the database and return the state.

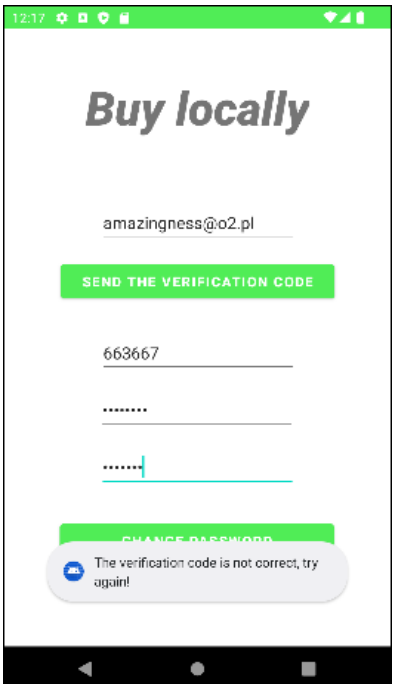
Then I proceeded to test this new functionality. First, I registered a new account with an old password. But then, I forgot what the password was, so I wanted to reset it.

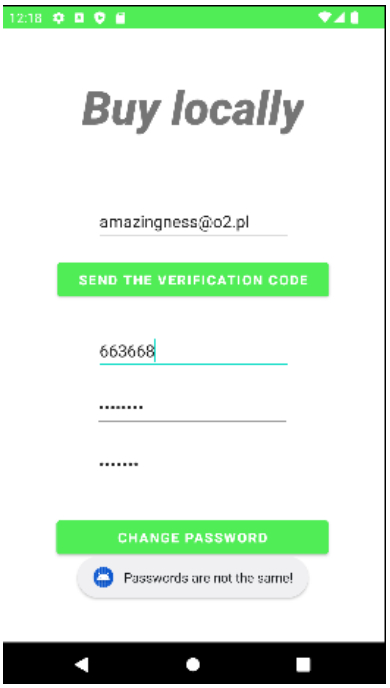


I then received an email with the verification code.

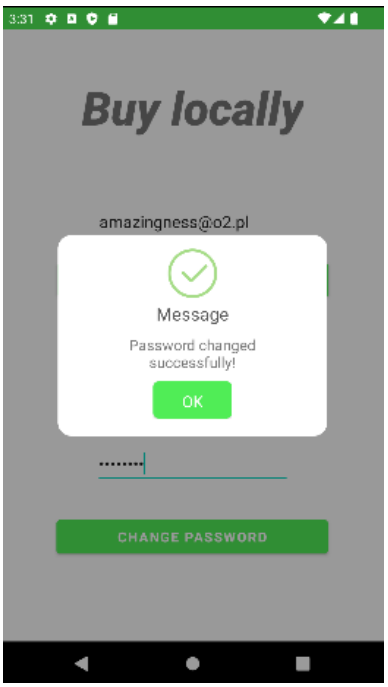


When I put in a wrong verification code or passwords do not meet the requirements, then I cannot change the password, which is the correct and expected behaviour.





When all details are correct, a success message is displayed and the record in the database amended. Then I am redirected to the login page and I can log in with the new password, which takes me to the home page, which is empty for now.



Testing table

Test number	Process tested	Input	Type of input	Expected output	Actual output
1	Splash screen	Opening the app	N/A	Splash screen animation playing for 3 seconds then redirecting to login	Correct
2	Logging in	Correct email and password	Normal	Success, log in the user	Error
3	Logging in	Incorrect email address	Erroneous	Error, does not log in	Success
4	Logging in	Incorrect password	Erroneous	Error, does not log in	Success
5	Logging in	No input	Erroneous	Error, no details were given	Success
6	Forgotten password	Correct details	Normal	Success, store the new password	Success
7	Forgotten password	Incorrect details	Erroneous	Error, no change in password	Success
8	Registering	Password match	Normal	Success, add the new user to database	Success
9	Registering	Passwords do not match	Erroneous	Error, passwords have to be the same	Success
10	Registering	No input	Erroneous	Error, no details were given	Success
11	Registering	Wrong email	Erroneous	Error, display message account already exists	Success

Third prototype

In this prototype, I wanted to implement a basic map activity, with locating the user on the map. I started by adding location permissions in the manifest file and importing Google Maps. I needed to add access to both coarse location and fine location, because fine location would not work without the coarse location access. Without these permissions, I would not be able to use the user's location and locate them on the map.

```
6       <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
7       <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

```
51      implementation 'com.google.android.gms:play-services-maps:18.1.0'
```

Then I created a new activity, which I called MapsActivity. It is going to be responsible for drawing the markers on the map. I have to be able to display markers on the map as they indicate where each producer is located.

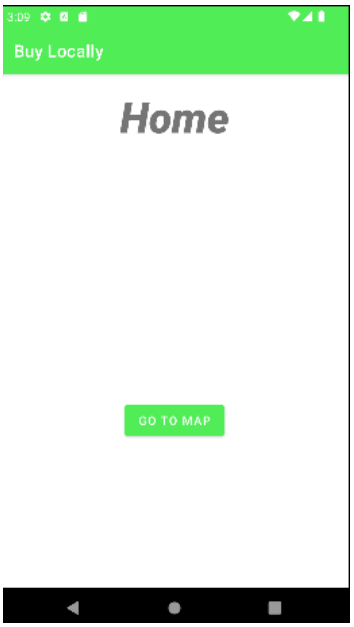
```
1   package com.example.firstprototype;
2
3   import ...
28
29  public class MapsActivity extends AppCompatActivity implements OnMapReadyCallback,
30          GoogleMap.OnMyLocationButtonClickListener, GoogleMap.OnMyLocationClickListener{
31
32
33      @Override
34      protected void onCreate(Bundle savedInstanceState) {
35          super.onCreate(savedInstanceState);
36          setContentView(R.layout.activity_maps);
37
38          // Get a handle to the fragment and register the callback.
39          SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager()
40                  .findFragmentById(R.id.map);
41          mapFragment.getMapAsync( callback: this);
42
43
44      }
45
46      @SuppressLint("MissingPermission")
47      @Override
48      public void onMapReady(GoogleMap googleMap) {
49
50
51          googleMap.setMyLocationEnabled(true);
52          googleMap.setOnMyLocationButtonClickListener(MapsActivity.this);
53          googleMap.setOnMyLocationClickListener(MapsActivity.this);
54
55
56          LatLng sydney = new LatLng( latitude: -33.852,  longitude: 151.211);
57          googleMap.addMarker(new MarkerOptions()
58                  .position(sydney)
59                  .title("Marker in Sydney"));
60          googleMap.moveCamera(CameraUpdateFactory.newLatLng(sydney));
61          googleMap.moveCamera(CameraUpdateFactory.zoomTo(15));
```

```
62
63
64
65      }
66
67      @Override
68      public boolean onMyLocationButtonClick() {
69          Toast.makeText( context: MapsActivity.this,  text: "Moving to your current location!", Toast.LENGTH_SHORT).show();
70          return false;
71      }
72
73      @Override
74      public void onMyLocationClick(@NonNull Location location) {
75          Toast.makeText( context: MapsActivity.this, text: "Current location:\n" + location, Toast.LENGTH_SHORT).show();
76      }
```

For now, the layout file for this activity is just the Google Maps fragment, which just covers the whole screen for now.

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <fragment xmlns:android="http://schemas.android.com/apk/res/android"
3       xmlns:map="http://schemas.android.com/apk/res-auto"
4       android:name="com.google.android.gms.maps.SupportMapFragment"
5       android:id="@+id/map"
6       android:layout_width="match_parent"
7       android:layout_height="match_parent"
8       map:uiZoomControls="true"
9       map:uiRotateGestures="true"
10      />
```

I also created a button to redirect from the dashboard to the map to allow for easy access to the map. For now, it is just going to be in the middle of the screen for testing purposes. I will later move it to a more sensible location on the screen.



But when I clicked the button to go to the map, the application crashed. I looked into the log console, and I found the following error.

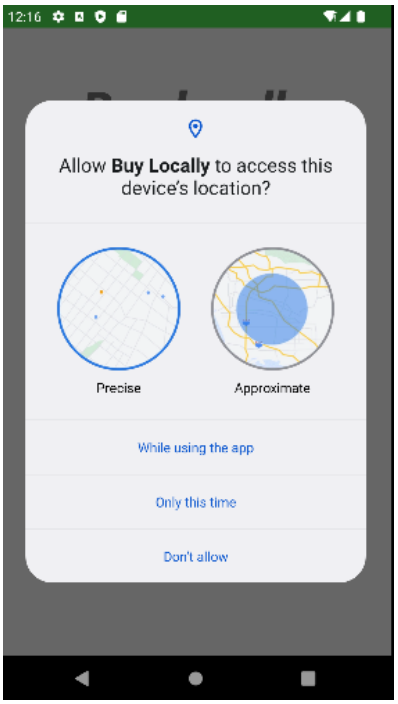
```
E FATAL EXCEPTION: main
  Process: com.example.firstprototype, PID: 13595
  java.lang.SecurityException: my location requires permission ACCESS_FINE_LOCATION or ACCESS_COARSE_LOCATION
      at com.google.maps.api.android.libб.impl.bj.J(:com.google.android.gms.dynamite_mapsdynamite@221514104@22.15.14
      at com.google.android.gms.maps.internal.i.aX(:com.google.android.gms.dynamite_mapsdynamite@221514104@22.15.14
      at eb.onTransact(:com.google.android.gms.dynamite_mapsdynamite@221514104@22.15.14 (190800-0):4)
      at android.os.Binder.transact(Binder.java:1064)
      at com.google.android.gms.internal.maps.zza.zzc(com.google.android.gms:play-services-maps@@18.1.0:2)
      at com.google.android.gms.maps.internal.zzg.setMyLocationEnabled(com.google.android.gms:play-services-maps@@18
      at com.google.android.gms.maps.GoogleMap.setMyLocationEnabled(com.google.android.gms:play-services-maps@@18.1.
      at com.example.firstprototype.MapsActivity.onMapReady(MapsActivity.java:45)
      at com.google.android.gms.maps.zzat.zzb(com.google.android.gms:play-services-maps@@18.1.0:1)
      at com.google.android.gms.maps.internal.zzar.zza(com.google.android.gms:play-services-maps@@18.1.0:6)
      at com.google.android.gms.internal.maps.zzb.onTransact(com.google.android.gms:play-services-maps@@18.1.0:3)
      at android.os.Binder.transact(Binder.java:1064)
```

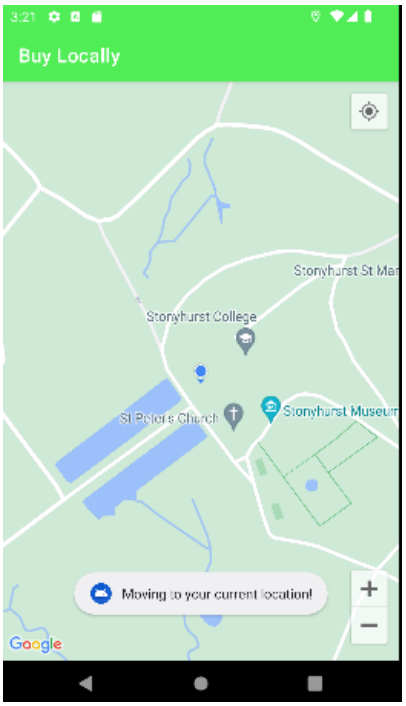
This means the app doesn't have permissions to use the location of the user. I thought the app gets the permission just by including it in the manifest file. However, this is not true and the app has to get this permission straight from the user. To resolve this issue, I added the following code to the MapActivity file to check if the user has given the permission, and if not, the app will ask the user to give the permission of location. I used the requestPermissions function to request the permission of access to the fine (accurate) location.

```
38          // check if the app has required permissions
39          if(!canAccessLocation()){
40              requestPermissions(new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, requestCode: 1337);
41          }
```

```
82
83          // helper functions
84          private boolean canAccessLocation() {
85              return(hasPermission(Manifest.permission.ACCESS_FINE_LOCATION));
86          }
87
88          private boolean hasPermission(String perm) {
89              return(PackageManager.PERMISSION_GRANTED==checkSelfPermission(perm));
90          }
91      }
```

Now, when I try to use the map, the app detects that the permission has not been granted, and asks the user to allow access to the device's location. After I agree, I can log in again and then use the map. When I first see the map, I see the test marker, which I set for Sidney. I can move around, use the buttons to zoom in and out. When I click the "locate me" button, the map moves to my current location, which is Stonyhurst College.





Now I can work on adding markers for the producers. To start adding producers, I first have to add a producer model class, similar to the customer model. I will create a new file in the project, defining the producer model class.

Testing table

Test number	Process tested	Input	Type of input	Expected output	Actual output
1	Splash screen	Opening the app	N/A	Splash screen animation playing for 3 seconds then redirecting to login	Correct
2	Logging in	Correct email and password	Normal	Success, log in the user	Success
3	Logging in	Incorrect email address	Erroneous	Error, does not log in	Success
4	Logging in	Incorrect password	Erroneous	Error, does not log in	Success
5	Logging in	No input	Erroneous	Error, no details were given	Success
6	Forgotten	Correct	Normal	Success, store	Success

	password	details		the new password	
7	Forgotten password	Incorrect details	Erroneous	Error, no change in password	Success
8	Registering	Password match	Normal	Success, add the new user to database	Success
9	Registering	Passwords do not match	Erroneous	Error, passwords have to be the same	Success
10	Registering	No input	Erroneous	Error, no details were given	Success
11	Registering	Wrong email	Erroneous	Error, display message account already exists	Success

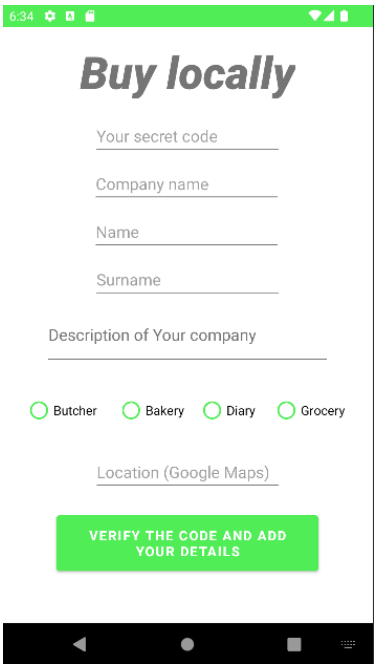
Fourth prototype

To start off this prototype, I created the producer model class. It is similar in structure to the customer model class. However, it has more attributes, which are ID, company name, first name, surname, description, type, and location. This class will allow me to easily refer to producers as objects with all necessary attributes defined.

```
package com.example.firstprototype;

public class ProducerModel {

    // declare all attributes
    private int id;
    private String companyName;
    private String firstName;
    private String surname;
    private String description;
    private String type;
    private String location;

    // constructors
    public ProducerModel(int id, String companyName, String firstName,
                         String surname, String description, String type, String location){
        this.id = id;
        this.companyName = companyName;
        this.firstName = firstName;
        this.surname = surname;
        this.description = description;
        this.type = type;
        this.location = location;
    }
```

Below that, I also implemented a toString method and all getters and setters. Then I created a new activity—producer dashboard, which enables the producers to put in their secret

code, and then they can add their company with all the necessary details. Below is the design of this activity.



Then I coded the activity producer dashboard which will display the information about the given producer. It takes all the input information, validates it, and calls a function to store them in the database.

```
package com.example.firstprototype;

import ...

public class ProducerDashboard extends AppCompatActivity {

    // references to all buttons and other controls on the layout
    Button btn_submit;
    EditText et_secretCode, et_companyName, et_firstName, et_surname, et_description, et_location;
    RadioButton rb_butcher, rb_bakery, rb_diary, rb_grocery;

    // define the secret code
    private final String SecretCode = "654321";

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_producer_dashboard);

        // hide the action bar
        if (getSupportActionBar() != null) {
            getSupportActionBar().hide();
        }

        // locate all elements on the screen
        btn_submit = findViewById(R.id.addProducer);
        et_secretCode = findViewById(R.id.secretCode);
        et_companyName = findViewById(R.id.companyName);
        et_firstName = findViewById(R.id.firstName);
```

```
        et_surname = findViewById(R.id.surname);
        et_description = findViewById(R.id.description);
        et_location = findViewById(R.id.location);
        rb_butcher = findViewById(R.id.butcher);
        rb_bakery = findViewById(R.id.bakery);
        rb_diary = findViewById(R.id.diary);
        rb_grocery = findViewById(R.id.grocery);

        // set a listener for the submit button
        btn_submit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) { addDetails(); }
        });
    }

    private void addDetails(){
        // get the string values of all elements
        String str_secret = et_secretCode.getText().toString();
        String str_company = et_companyName.getText().toString();
        String str_firstName = et_firstName.getText().toString();
        String str_surname = et_surname.getText().toString();
        String str_description = et_description.getText().toString();
        String str_location = et_location.getText().toString();
        String type = "";
        // a marker to know if an option has been chosen
        boolean selectedType = false;


        // check if the secret code is correct, which for now is 654321
```

```
        if(!str_secret.equals(SecretCode)){                                    9
            Toast.makeText( context: this,  text: "Incorrect secret code!", Toast.LENGTH_SHORT).show();
        } else if(str_company.length() < 3){
            Toast.makeText( context: this,  text: "Company name must be at least 3 characters long!", Toast.LENGTH_SHORT).show();
        } else if(str_firstName.length() < 2 || str_surname.length() < 3){
            Toast.makeText( context: this,  text: "Enter Your name and surname!", Toast.LENGTH_SHORT).show();
        } else if(str_description.length() < 10){
            Toast.makeText( context: this,  text: "Description must be at least 10 characters long!", Toast.LENGTH_SHORT).show();
        } else {
            // check the buttons to check which one has been selected and if any has been selected
            if (rb_butcher.isChecked()) {
                type = rb_butcher.getText().toString();
                selectedType = true;
            } else if (rb_bakery.isChecked()) {
                type = rb_bakery.getText().toString();
                selectedType = true;
            } else if (rb_diary.isChecked()) {
                type = rb_diary.getText().toString();
                selectedType = true;
            } else if (rb_grocery.isChecked()) {
                type = rb_grocery.getText().toString();
                selectedType = true;
            }
            // check if any type was selected
            if (!selectedType) {
                Toast.makeText( context: this,  text: "Select a type!", Toast.LENGTH_SHORT).show();
            } else if (!checkValidLocation(str_location)) {
                // if wrong format of location then stop
            } else {
```

```
                // create a producer object
                ProducerModel producer = new ProducerModel( id: -1, str_company, str_firstName, str_surname,
                str_description, type, str_location);

                // add the producer to the database
                DBHelper databaseHelper  = new DBHelper( context: ProducerDashboard.this);
                boolean success = databaseHelper.addProducer(producer);

                // signal that information has been added successfully
                new SweetAlertDialog( context: ProducerDashboard.this, SweetAlertDialog.SUCCESS_TYPE)
                        .setTitleText("Message")
                        .setContentText("Updated info successfully!")
                        .setConfirmText("OK")
                        .setConfirmClickListener(new SweetAlertDialog.OnSweetClickListener() {
                            @Override
                            public void onClick(SweetAlertDialog sweetAlertDialog) {
                                final Intent i = new Intent( packageContext: ProducerDashboard.this, LoginUser.class);
                                startActivity(i);
                            }
                        })
                        .show();
            }
        }
    }

    // helper function to check if the location is in correct format
    boolean checkValidLocation(String location){
```

```
        // using try to catch conversion errors
        try {
            // check for comma and if it is not the last character
            if(location.indexOf(",")==location.length()-1){
                Toast.makeText( context: this,  text: "Enter a valid location!", Toast.LENGTH_SHORT).show();
                return false;
            }
            // using split function to get the two parts of ths string and then converting it to double
            String[] parts = location.split( regex: ",");
            double longitude = Double.valueOf(parts[0]);
            double latitude = Double.valueOf(parts[1]);
            System.out.println(longitude);
            System.out.println(latitude);
            return true;
        } catch (NumberFormatException e) {
            // if conversion failed, then location is not valid
            Toast.makeText( context: this,  text: "Enter a valid location!", Toast.LENGTH_SHORT).show();
            return false;
        }

    }
}
```

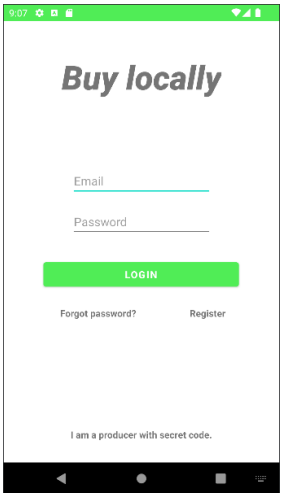
Next, I added code to the database helper to create a new table for producers. First, I define all important string variables as final, and I will use them in the SQL query to avoid hard-written code. Thanks to that, the program will be less prone to errors. Then I implemented the function to add a new producer, which is really similar to adding a customer. It just has more arguments and of course writes the data into the producers table. I am again using a contentvalues object to put the details into the format understandable for the SQLite database.

```
// for the producers
public static final String PRODUCER_TABLE = "PRODUCER_TABLE";
public static final String COLUMN_COMPANY_NAME = "COMPANY";
public static final String COLUMN_DESCRIPTION = "DESCRIPTION";
public static final String COLUMN_TYPE = "TYPE";
public static final String COLUMN_LOCATION = "LOCATION";
```

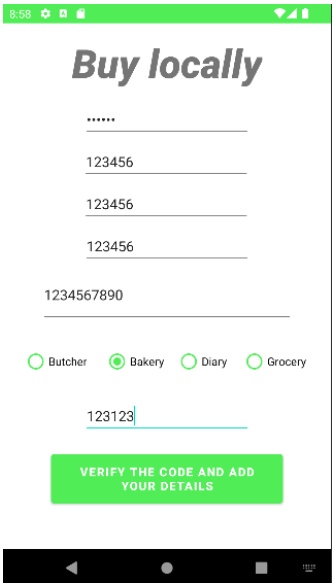
```
String createProducerTableStatement = "CREATE TABLE " + PRODUCER_TABLE +
        " (" + COLUMN_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, "
        + COLUMN_COMPANY_NAME + " TEXT, "
        + COLUMN_FIRSTNAME + " TEXT, "
        + COLUMN_SURNAME + " TEXT, "
        + COLUMN_DESCRIPTION + " TEXT, "
        + COLUMN_TYPE + " TEXT, "
        + COLUMN_LOCATION + " TEXT)";

db.execSQL(createProducerTableStatement);
```

```java
//---------------- for producers --------------------//

// creating the company record
public boolean addProducer(ProducerModel producer){

    // get the access to the database
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues cv = new ContentValues();

    // insert the data to content values
    cv.put(COLUMN_COMPANY_NAME, producer.getCompanyName());
    cv.put(COLUMN_FIRSTNAME, producer.getFirstName());
    cv.put(COLUMN_SURNAME, producer.getSurname());
    cv.put(COLUMN_DESCRIPTION, producer.getDescription());
    cv.put(COLUMN_TYPE, producer.getType());
    cv.put(COLUMN_LOCATION, producer.getLocation());

    // add the record to the database and get the result
    long insert = db.insert(PRODUCER_TABLE,  nullColumnHack: null, cv);
    db.close();

    // return the correct boolean result
    if(insert == -1){
        return false;
    }
    return true;
}
```

And finally, I added a button to the home page which redirects to the producer dashboard. This button takes the user to the producer register page, where they can input the secret code and all details about them to add their company into the database and make it visible for customers.



Then, I went over to testing the new activity. For this basic test, I use some placeholder data just to see if the activity understands it correctly and adds the record to the database.



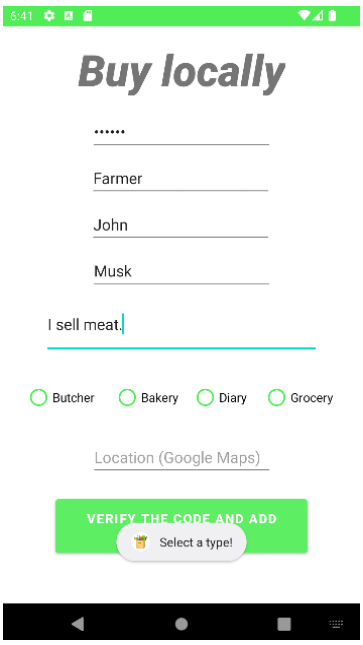
I put in some testing data, and after clicking the verify button, the app crashed and the following error occurred.

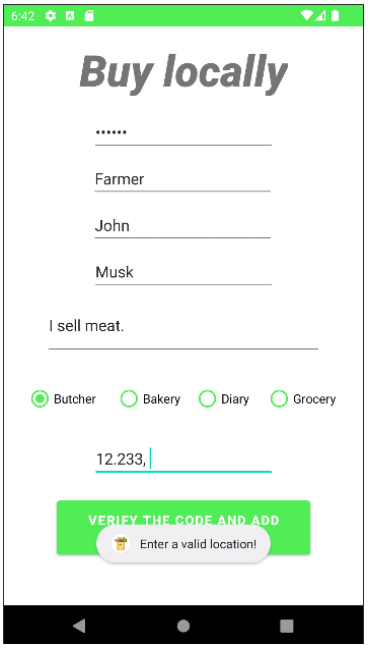
```
E FATAL EXCEPTION: main
  Process: com.example.firstprototype, PID: 15874
  java.lang.ArrayIndexOutOfBoundsException: length=1; index=1
      at com.example.firstprototype.ProducerDashboard.checkValidLocation(ProducerDashboard.java:139)
      at com.example.firstprototype.ProducerDashboard.addDetails(ProducerDashboard.java:99)
      at com.example.firstprototype.ProducerDashboard.access$000(ProducerDashboard.java:17)
      at com.example.firstprototype.ProducerDashboard$1.onClick(ProducerDashboard.java:54)
      at android.view.View.performClick(View.java:7441)
      at com.google.android.material.button.MaterialButton.performClick(MaterialButton.java:1131)
      at android.view.View.performClickInternal(View.java:7418)
      at android.view.View.access$3700(View.java:835)
      at android.view.View$PerformClick.run(View.java:28676)
      at android.os.Handler.handleCallback(Handler.java:938)
      at android.os.Handler.dispatchMessage(Handler.java:99)
      at android.os.Looper.loopOnce(Looper.java:201)
      at android.os.Looper.loop(Looper.java:288)
      at android.app.ActivityThread.main(ActivityThread.java:7839) <1 internal line>
      at com.android.internal.os.RuntimeInit$MethodAndArgsCaller.run(RuntimeInit.java:548)
```

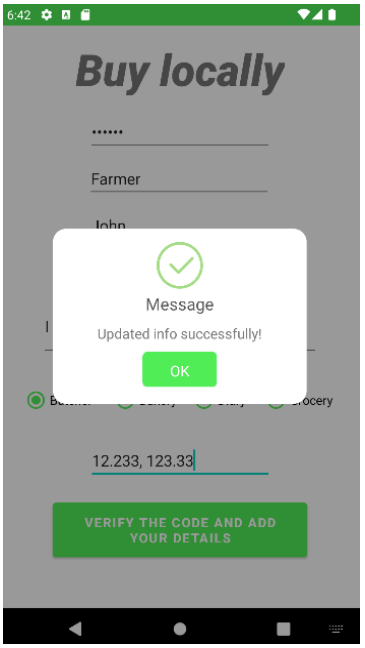
I found out the error was that my program would allow a string with no comma, which then crashes the split function, because there is no comma in the string. A fix of that is checking if the string contains a comma, and if not then do not proceed. This error had be fixed, as if the producer puts in incorrect location data, they won't be displayed on the map and the app would crash.

```
try {
    // check for comma and if it is not the last character
    if(!location.contains(",") || location.indexOf(",")==location.length()-1){
        Toast.makeText( context: this,  text: "Enter a valid location!", Toast.LENGTH_SHORT).show();
        return false;
    }
```

Then, I tested the producer registration page. When some details are missing, an appropriate message is shown to describe what is missing. When all details are in the correct format, a success message is displayed and the producer is added to the database.







Testing table

Test number	Process tested	Input	Type of input	Expected output	Actual output
1	Adding a producer	Correct secret code, company name, description, type selected and location format correct	Normal	Success, add the producer to the database	Success
2	Adding a producer	Incorrect secret code	Erroneous	Error, does not add any details to the database	Success
3	Adding a producer	Any incorrect input format	Erroneous	Error, does not add any details to the database	Success, correct error toast displayed
4	Adding a producer	No type selected or no details	Erroneous	Error, the necessary information was not given	Success

Fifth prototype

In this prototype, I wanted to implement presenting all the producers' markers on the map. I started by a test run of the current map setup. However, when I clicked the "go to map" button, the app crashed. The following error occurred.

Such error cannot exist in the app because it limits the usability. I am not sure what caused it, but I found the solution, aided by information from this thread. The issue was that I did not include one of the necessary permissions—access to the network state. After adding the following line to the manifest file, the map opened correctly.

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Next, I added code to loop through all the producers and display them on the map, with a marker of an appropriate colour. All the producers have to be displayed on the map for the users to discover them and see them

```
99          DBHelper databaseHelper = new DBHelper( context: MapsActivity.this);
100         List<ProducerModel> allProducers = databaseHelper.getAllProducers();
101         for(ProducerModel producer : allProducers){
102             String location = producer.getLocation();
103             String[] parts = location.split( regex: ",");
104             double latitude = Double.valueOf(parts[0]);
105             double longitude = Double.valueOf(parts[1]);
106             LatLng where = new LatLng(latitude, longitude);
107
108             float colour = 0;
109             if (producer.getType().equals("Butcher")) {
110                 colour = BitmapDescriptorFactory.HUE_RED;
111             } else if (producer.getType().equals("Grocery")) {
112                 colour = BitmapDescriptorFactory.HUE_GREEN;
113             } else if (producer.getType().equals("Bakery")) {
114                 colour = BitmapDescriptorFactory.HUE_YELLOW;
115             } else {
116                 // it has to diary
117                 colour = BitmapDescriptorFactory.HUE_CYAN;
118             }
119
120             Marker marker = googleMap.addMarker(new MarkerOptions()
121                     .position(where)
122                     .title(producer.getCompanyName())
123                     .snippet(producer.getFirstName() + " " + producer.getSurname())
124                     .icon(BitmapDescriptorFactory.defaultMarker(colour)));
125
126             markersList.add(marker);
127         }
```

Then I created a function to return a list of all producers in the system, to display all of them on the map. I coded the following function in the database helper class. The function uses a

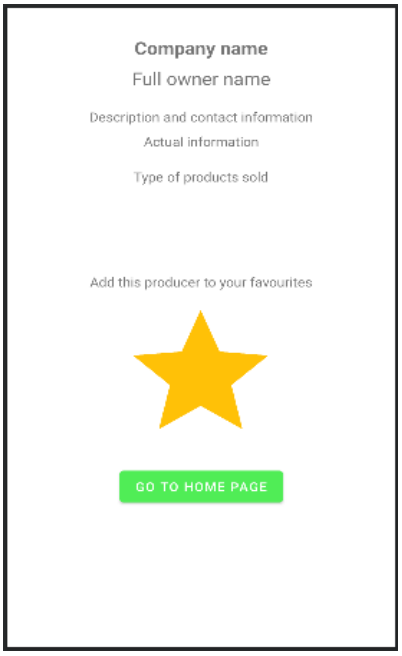
select statement with a * to select all records from the database. Then it takes the details out in String variables.

```
    // get a list of all producers
    public List<ProducerModel> getAllProducers(){
        List<ProducerModel> retList = new ArrayList<>();
        // get the data from the database
        String selectStatement = "SELECT * FROM " + PRODUCER_TABLE;

        // get access to the database
        SQLiteDatabase db = this.getReadableDatabase();
        // do a query to search the database
        Cursor cursor = db.rawQuery(selectStatement,  selectionArgs: null);
        if(cursor.moveToFirst()){
            // loop through the result and create new objects and add them to the list
            do {
                // convert all cursor values to their types
                int id = cursor.getInt( i: 0);
                String companyName = cursor.getString( i: 1);
                String firstName = cursor.getString( i: 2);
                String surname = cursor.getString( i: 3);
                String description = cursor.getString( i: 4);
                String type = cursor.getString( i: 5);
                String location = cursor.getString( i: 6);

                ProducerModel newProducer = new ProducerModel(id, companyName, firstName, surname, description, type, location);
                retList.add(newProducer);

            } while (cursor.moveToNext());
```

```
        } else {
            // the list will just be empty - no matching results
        }
        // close both the database and the cursor
        cursor.close();
        db.close();
        return retList;
    }
```

This function returns a list with all producers which are currently in the system, which is needed to be able to display them on the map.

Next, I created a new activity—producer profile activity, which will display all the information about the given producer. The layout file looks as follows.



So when the user clicks the info window on a given marker with the producer, they should be redirected to this page with the correct producer information. Then the activity can retrieve this producer from the database and display all the info on the screen. To share the information between different activities, I am using shared preferences, to store the data and to be able to easily retrieve it when needed.

```
public class ProducerProfile extends AppCompatActivity {

    // define all necessary variables
    TextView companyName, fullName, description, type, favouritesText;
    ImageView star;
    Button btn_goToHome;
    CustomerModel CurrentCustomer;
    ProducerModel CurrentProducer;
    private boolean isItFavourite;
    private AlphaAnimation fadeIn = new AlphaAnimation(0.0f , 1.0f);
    private AlphaAnimation fadeOut = new AlphaAnimation(1.0f , 0.0f);
    private int LastKnownColour;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_producer_profile);
        if(getSupportActionBar() != null) {
            getSupportActionBar().setSubtitle("Producer profile");
        }
        // reference all elements on the screen
        companyName = findViewById(R.id.profileCompanyName);
        fullName = findViewById(R.id.fullName);
        description = findViewById(R.id.descriptionContact);
        type = findViewById(R.id.productType);
        favouritesText = findViewById(R.id.favouritesText);
        star = findViewById(R.id.star);
        btn_goToHome = findViewById(R.id.goToHome);

        // use shared preferences to know what producer data to fetch
```

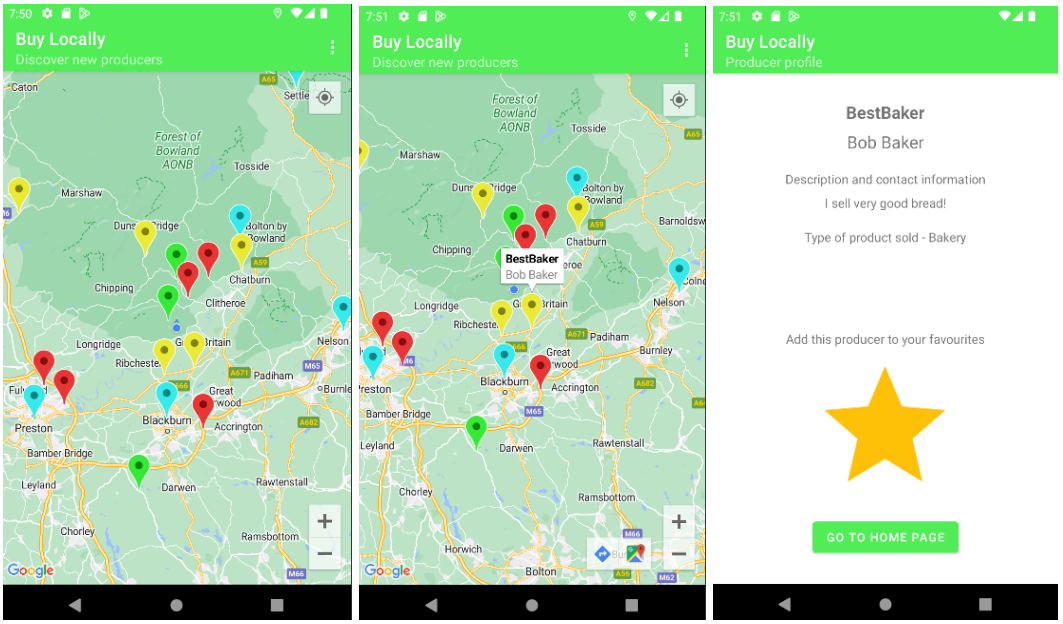
```
        SharedPreferences sharedPreferences = getSharedPreferences( name: "MyShared", MODE_PRIVATE);
        String str_companyName = sharedPreferences.getString( s: "companyName", s1: "");
        String str_email = sharedPreferences.getString( s: "email", s1: "");

        // fetch the producer from the database
        DBHelper databaseHelper = new DBHelper( context: ProducerProfile.this);
        ProducerModel producer = databaseHelper.getProducerByNameOrID(str_companyName,  ID: -1).get(0);
        CustomerModel customer = databaseHelper.getAllMatching(str_email).get(0);

        CurrentCustomer = customer;
        CurrentProducer = producer;

        // set the appropriate text on the screen
        companyName.setText(producer.getCompanyName());
        fullName.setText(producer.getFirstName() + " " + producer.getSurname());
        description.setText(producer.getDescription());
        type.setText("Type of product sold - " + producer.getType());

        LastKnownColour = ContextCompat.getColor( context: this, R.color.yellow);
        isItFavourite = false;
        // check if the producer is in favourites, if yes then amend the text on the screen
        List<Integer> allFavourites = databaseHelper.getAllFavourites(customer);
        if(allFavourites.contains(producer.getId())){
            favouritesText.setText("Remove this producer from your favourites");
            LastKnownColour = ContextCompat.getColor( context: this, R.color.red);
            star.setColorFilter(ContextCompat.getColor( context: this, R.color.red));
            isItFavourite = true;
        }
```

```
        star.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                // functions to be implemented
                if(!isItFavourite) {
                    addToFavourites(CurrentCustomer, CurrentProducer);
                } else {
                    removeFromFavourites(CurrentCustomer, CurrentProducer);
                }
            }
        });

        btn_goToHome.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                final Intent i = new Intent( packageContext: ProducerProfile.this, HomeActivity.class);
                startActivity(i);
            }
        });

    }
```

This activity also contains variables which I will use later and functions which I will have to implement later, when I have a way to store favourite producers of every producer in the database. Then I needed a way to call this activity, and I wanted to call it when the info window is clicked, and add the necessary data to shared preferences. This function does this.

```
    @Override
    public void onInfoWindowClick(@NonNull Marker marker) {
        // Toast.makeText(MapsActivity.this, "Info window clicked!", Toast.LENGTH_SHORT).show();
        SharedPreferences sharedPreferences = getSharedPreferences( name: "MyShared", MODE_PRIVATE);
        SharedPreferences.Editor editor = sharedPreferences.edit();
        editor.putString( s: "companyName", marker.getTitle());
        editor.apply();

        Intent i = new Intent( packageContext: MapsActivity.this, ProducerProfile.class);
        startActivity(i);
    }
```

Now, all the producers should be visible on the map and if the info window is clicked, the user is redirected to the producer profile where the correct details are shown.



As the last step in this prototype, I added a quick way of filling up the database of producers, purely for testing reasons, especially on other devices rather than my own. I added the following to the producer dashboard activity code.

```
74          // for testing on other devices purpose
75          if(str_company.equals("quickstart")){
76              addABunchOfProducers();
77              Toast.makeText( context: this,  text: "Added a bunch of producers for testing!", Toast.LENGTH_SHORT).show();
78          }
```

```
159     // helper function to enable testing on other devices
160     private void addABunchOfProducers(){
161         List<ProducerModel> producersList = new ArrayList<>();
162         producersList.add(new ProducerModel( id: -1,  companyName: "BestBaker",  firstName: "Bob",  surname: "Baker"
163                 ,  description: "I sell very good bread!",  type: "Bakery",  location: "53.808669, -2.443445"));
164         producersList.add(new ProducerModel( id: -1,  companyName: "WorstBaker",  firstName: "Bob",  surname: "Filter"
165                 ,  description: "I sell very bad bread!",  type: "Bakery",  location: "53.802192, -2.492404"));
```

And after all producers' declarations, I add them to the database.

```
206             DBHelper databaseHelper = new DBHelper( context: ProducerDashboard.this);
207             for(ProducerModel producer : producersList){
208                 databaseHelper.addProducer(producer);
209             }
210         }
```

Testing table

Test number	Process tested	Input	Type of input	Expected output	Actual output
1	Info window on the map	Once clicked	Normal	Opening the producer profile page with the correct details	Success
2	Empty list element	Once clicked	Normal	Opening the map to add producers, because the list is empty	Not yet implemented
3	Star in the producer profile	Once clicked	Normal	Adding the producer to favourites and displaying them later on the list on the main page	Success
4	List element on the main page	Once clicked	Normal	Opening the producer profile page with the correct details	Not yet implemented
5	Sort options	Once clicked	Normal	Sort the favourites list accordingly	Not yet implemented
6	Go to map button	Once clicked	Normal	Takes the user to the map	Success

Sixth prototype

In this last prototype, I will implement all the favourite producers' features, which are to create the link table in the database, enable adding the producers to favourites, and then displaying them in a list on the main page. To achieve all of this,I first created a new table in the database—link table, which will store all connections between users and their favourite producers.

```
71          String createLinkTableStatement = "CREATE TABLE LINK_TABLE(" +
72                  "CUSTOMERID INTEGER, " +
73                  "PRODUCERID INTEGER, " +
74                  "FOREIGN KEY(CUSTOMERID) REFERENCES CUSTOMER_TABLE(ID), " +
75                  "FOREIGN KEY(PRODUCERID) REFERENCES PRODUCER_TABLE(ID), " +
76                  "PRIMARY KEY(CUSTOMERID, PRODUCERID))";
77
78          db.execSQL(createLinkTableStatement);
```

Then I called two functions, one to add the producer to favourites and another one to remove them from favourites.

```
82      star.setOnClickListener(new View.OnClickListener() {
83          @Override
84          public void onClick(View view) {
85              if(!isItFavourite) {
86                  addToFavourites(CurrentCustomer, CurrentProducer);
87              } else {
88                  removeFromFavourites(CurrentCustomer, CurrentProducer);
89              }
90          }
91      });
```

I added these functions to the same file—producer profile. The add to favourites function. I added some fancy and maybe not-so necessary animation to the star when the user clicks them. Essentially, it is just an indicator that the preference of the producer has changed. I also added a dynamic colour change to the star to make it more appealing for the users.

```
103     private void addToFavourites(CustomerModel customer, ProducerModel producer){
104         // call the function from database helper
105         DBHelper databaseHelper = new DBHelper( context: ProducerProfile.this);
106         boolean success = databaseHelper.addToFavourites(customer, producer);
107         isItFavourite = true;
108         // animate the star and text
109         star.animate().setDuration(2000);
110         star.animate().rotationBy(1080 - star.getRotation());
111         star.animate().rotationYBy(360 - star.getRotationY());
112         favouritesText.startAnimation(fadeOut);
113         favouritesText.setText("Remove this producer from your favourites");
114         favouritesText.startAnimation(fadeIn);
115         fadeOut.setDuration(1000);
116         fadeOut.setFillAfter(true);
117         fadeIn.setDuration(2000);
118         fadeIn.setFillAfter(true);
119         // for smooth colour change
120         int colourFrom = LastKnownColour;
121         int colourTo = ContextCompat.getColor( context: this, R.color.red);
122         ValueAnimator valueAnimator = ValueAnimator.ofObject(new ArgbEvaluator(), colourFrom, colourTo);
123         valueAnimator.setDuration(2000);
124         valueAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
125             @Override
126             public void onAnimationUpdate(ValueAnimator valueAnimator) {
127                 LastKnownColour = (int) valueAnimator.getAnimatedValue();
128                 star.setColorFilter(LastKnownColour);
129             }
130         });
131         valueAnimator.start();
132     }
```

And the remove from favourites function, which is very similar and essentially reverses the process. It does all animations and colour gradient change as before, just in reverse order.

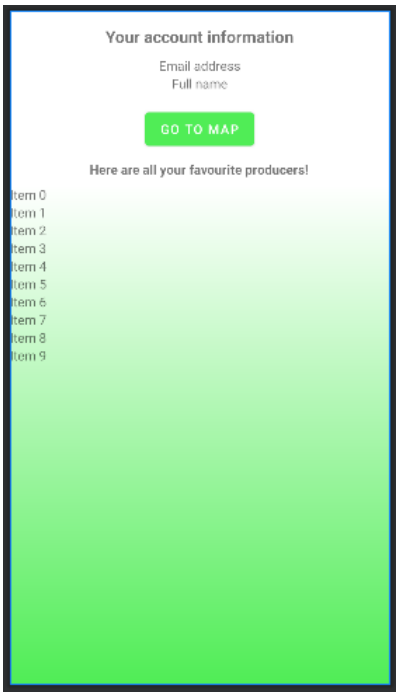
```
134     private void removeFromFavourites(CustomerModel customer, ProducerModel producer){
135         // call the function from the database helper
136         DBHelper databaseHelper = new DBHelper( context: ProducerProfile.this);
137         boolean success = databaseHelper.removeFromFavourites(customer, producer);
138         isItFavourite = false;
139         // animate the star and the text
140         star.animate().setDuration(2000);
141         star.animate().rotationBy(-1080 - star.getRotation());
142         star.animate().rotationYBy(-360 - star.getRotationY());
143         favouritesText.startAnimation(fadeOut);
144         favouritesText.setText("Add this producer to your favourites");
145         favouritesText.startAnimation(fadeIn);
146         fadeOut.setDuration(1000);
147         fadeOut.setFillAfter(true);
148         fadeIn.setDuration(2000);
149         fadeIn.setFillAfter(true);
150         // for smooth colour change
151         int colourFrom = LastKnownColour;
152         int colourTo = ContextCompat.getColor( context: this, R.color.yellow);
153         ValueAnimator valueAnimator = ValueAnimator.ofObject(new ArgbEvaluator(), colourFrom, colourTo);
154         valueAnimator.setDuration(2000);
155         valueAnimator.addUpdateListener(new ValueAnimator.AnimatorUpdateListener() {
156             @Override
157             public void onAnimationUpdate(ValueAnimator valueAnimator) {
158                 LastKnownColour = (int) valueAnimator.getAnimatedValue();
159                 star.setColorFilter(LastKnownColour);
160             }
161         });
162         valueAnimator.start();
163     }
```

Then I implemented the functions in the database helper, to actually record the changes in the database. Adding to favourites just inserts the correct information into the link table. Thanks to that, many users can have many different favourite producers and form a many-to-many relationship.

```
287  @   public boolean addToFavourites(CustomerModel customer, ProducerModel producer){
288
289          // get the access to the database
290          SQLiteDatabase db = this.getWritableDatabase();
291          ContentValues cv = new ContentValues();
292
293          // insert the data to content values
294          cv.put(COLUMN_CUSTOMER_ID, customer.getId());
295          cv.put(COLUMN_PRODUCER_ID, producer.getId());
296
297          // add the record to the database and get the result
298          long insert = db.insert(LINK_TABLE,  nullColumnHack: null, cv);
299          db.close();
300
301          // return the correct boolean result
302          if(insert == -1){
303              return false;
304          }
305          return true;
306      }
```

```
333  @   public boolean removeFromFavourites(CustomerModel customer, ProducerModel producer){
334          SQLiteDatabase db = this.getWritableDatabase();
335          boolean result = db.delete(LINK_TABLE,  whereClause: COLUMN_CUSTOMER_ID + "=" + customer.getId()
336                  + " AND " + COLUMN_PRODUCER_ID + "=" + producer.getId(),  whereArgs: null) > 0;
337          db.close();
338          return result;
339      }
```

Then, I proceeded to modify the home page, to incorporate a list of all producers which the current user has in their favourites. The list is implemented with a recycler view. I need a list to display all favourite producers of the currently signed in user. I created a RecyclerView, aided by this tutorial.



Then I implemented a new function in the database helper, which will fetch all producers which the given user added to their favourites. The function takes a customer as an argument, and returns a list of IDs of all the producers liked by the given customer.

```java
// get all producers which this user added to their favourites
public List<Integer> getAllFavourites(CustomerModel customer){
    List<Integer> retList = new ArrayList<>();

    // get the data from the database
    String selectStatement = "SELECT * FROM " + LINK_TABLE + " WHERE " + COLUMN_CUSTOMER_ID + " = '" + customer.getId() + "'";
    SQLiteDatabase db = this.getReadableDatabase();

    Cursor cursor = db.rawQuery(selectStatement, selectionArgs: null);
    if(cursor.moveToFirst()){
        // loop through the result and create new customers and add them to the list
        do {
            int customerID = cursor.getInt( i: 0);
            int producerID = cursor.getInt( i: 1);
            retList.add(producerID);
        } while (cursor.moveToNext());
    } else {
        // the list will just be empty - no matching results
    }
    // close both the database and the cursor
    cursor.close();
    db.close();
    return retList;
}
```

As this function returns a list of integers—IDs of producers, I need to modify the get producer function, to also accept an integer—ID—as its parameter.

```java
// get a list of all producers with given name or ID
public List<ProducerModel> getProducerByNameOrID(String checkedName, int ID){
    List<ProducerModel> retList = new ArrayList<>();
    String selectStatement = "";
    // decide which statement to use
    if(ID == -1) {
        // get the data from the database
        selectStatement = "SELECT * FROM " + PRODUCER_TABLE + " WHERE " + COLUMN_COMPANY_NAME + " = '" + checkedName + "'";
    } else {
        selectStatement = "SELECT * FROM " + PRODUCER_TABLE + " WHERE " + COLUMN_ID + " = '" + ID + "'";
    }
    // get access to the database
    SQLiteDatabase db = this.getReadableDatabase();

    // do a query to search the database
    Cursor cursor = db.rawQuery(selectStatement, selectionArgs: null);
    if(cursor.moveToFirst()){
        // loop through the result and create new customers and add them to the list
        do {
            int id = cursor.getInt( i: 0);
            String companyName = cursor.getString( i: 1);
            String firstName = cursor.getString( i: 2);
            String surname = cursor.getString( i: 3);
            String description = cursor.getString( i: 4);
            String type = cursor.getString( i: 5);
            String location = cursor.getString( i: 6);

            ProducerModel newProducer = new ProducerModel(id, companyName, firstName, surname, description, type, location);
            retList.add(newProducer);
        } while (cursor.moveToNext());
    } else {
        // the list will just be empty - no matching results
    }
    // close both the database and the cursor
    cursor.close();
    db.close();
    return retList;
}
```

Then I implemented and updated most of the main activity code. I needed to add the recycler view and the list of producers to be able to display it on the screen, in the recycler view. I already declared a menu, although I will be using later, to enable the customers to sort the list of favourite producers.

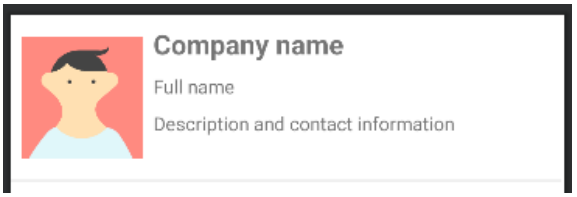
```
29      Button goToMap;
30      TextView accountEmail, fullName;
31      RecyclerView favouritesList;
32      RecyclerView.Adapter listAdapter;
33      RecyclerView.LayoutManager layoutManager;
34      Menu menu;
35      List<ProducerModel> allFavourites;
```

I needed to implement the new features to the main function of the main activity, to connect them to the main framework of the program.

```
38      @Override
39      protected void onCreate(Bundle savedInstanceState) {
40          super.onCreate(savedInstanceState);
41          setContentView(R.layout.activity_home);
42          if(getSupportActionBar() != null) {
43              getSupportActionBar().setSubtitle("Home");
44          }
45
46          goToMap = findViewById(R.id.goToMap);
47          accountEmail = findViewById(R.id.homeAccountEmail);
48          fullName = findViewById(R.id.homeFullName);
49          favouritesList = findViewById(R.id.listOfProducers);
50          // get the current user and find them and display info on the screen
51          SharedPreferences sharedPreferences = getSharedPreferences( name: "MyShared", MODE_PRIVATE);
52          String email = sharedPreferences.getString( s: "email", s1: "");
53          DBHelper databaseHelper = new DBHelper( context: HomeActivity.this);
54          CustomerModel found = databaseHelper.getAllMatching(email).get(0);
55          if(email != null){
56              accountEmail.setText("Email: " + found.getEmail());
57              fullName.setText("Welcome back " + found.getFirstName() + " " + found.getSurname() + "!");
58          }
59          // get all favourite producers' IDs
60          List<Integer> allFavouritesIDs = databaseHelper.getAllFavourites(found);
61          allFavourites = new ArrayList<>();
62          for(int id : allFavouritesIDs){
63              allFavourites.add(databaseHelper.getProducerByNameOrID( checkedName: null, id).get(0));
64          }
65          // if no favourite producers, add an empty item to the list
66          if(allFavourites.size() == 0){
```

```
67              allFavourites.add(new ProducerModel( id: 0, companyName: "Looks like the list is empty!",
68                      firstName: "", surname: "",
69                      description: "Add some producers to your favourites to see them here!", type: "none", location: ""));
70          }
71          // list management
72          favouritesList.setHasFixedSize(true);
73          layoutManager = new LinearLayoutManager( context: this);
74          favouritesList.setLayoutManager(layoutManager);
75
76          listAdapter = new RecycleViewAdapter(allFavourites, context: HomeActivity.this);
77          favouritesList.setAdapter(listAdapter);
78
79          goToMap.setOnClickListener(new View.OnClickListener() {
80              @Override
81              public void onClick(View view) {
82                  final Intent i = new Intent( packageContext: HomeActivity.this, MapsActivity.class);
83                  startActivity(i);
84              }
85          });
86
87      }
```

Then, I needed to implement the recycler view adapter, to use a prepared layout for an individual element of the list. So first I created a new layout file, for a singular element of the list.



After that, I created a new Java class - RecycleViewAdapter. It is responsible for displaying all producer cards in the recycler view list. Most of this code is solely needed because of the nature of lists in android studio.

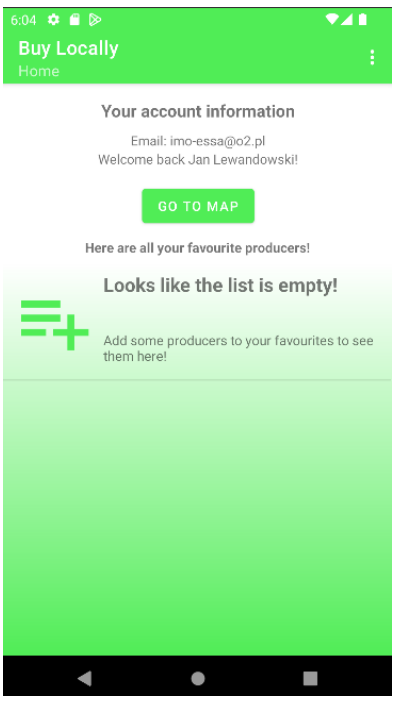
```java
public class RecycleViewAdapter extends RecyclerView.Adapter<RecycleViewAdapter.MyViewHolder> {

    List<ProducerModel> producersList;
    Context context;

    // constructor
    public RecycleViewAdapter(List<ProducerModel> producers, Context context) {
        this.producersList = producers;
        this.context = context;
    }

    @NonNull
    @Override
    public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
        View view = LayoutInflater.from(parent.getContext()).inflate(R.layout.one_list_item, parent, attachToRoot: false);
        MyViewHolder holder = new MyViewHolder(view);
        return holder;
    }
    // set all value in the card template with the correct text and image
    @Override
    public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
        ProducerModel currentProducer = producersList.get(position);
        String type = currentProducer.getType();
        if(type.equals("Butcher")){
            holder.iv_typePicture.setImageResource(R.drawable.meat);
        } else if(type.equals("Grocery")){
            holder.iv_typePicture.setImageResource(R.drawable.vegetable);
        } else if(type.equals("Bakery")){
            holder.iv_typePicture.setImageResource(R.drawable.bread);
        } else if(type.equals("none")){
```

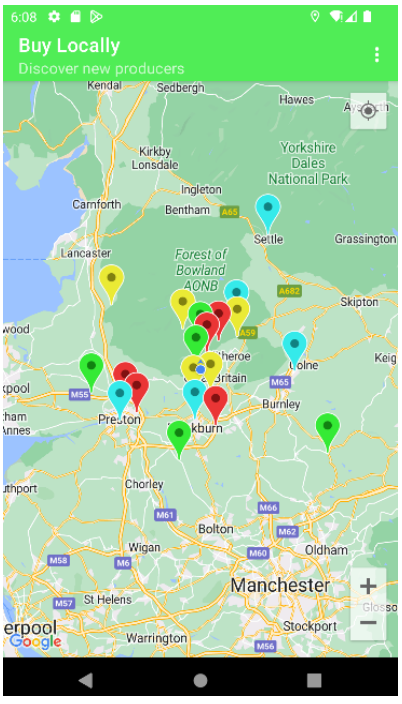
```java
            holder.iv_typePicture.setImageResource(R.drawable.type_none_image);
        } else {
            // must be diary
            holder.iv_typePicture.setImageResource(R.drawable.cheese);
        }
        holder.tv_companyName.setText(currentProducer.getCompanyName());
        holder.tv_fullName.setText(currentProducer.getFirstName() + " " + currentProducer.getSurname());
        holder.tv_description.setText(currentProducer.getDescription());

        // set a click listener and open producer profile (or map if the list is empty)
        holder.parentLayout.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View view) {
                if(!currentProducer.getType().equals("none")) {
                    SharedPreferences sharedPreferences = context.getSharedPreferences( s: "MyShared", MODE_PRIVATE);
                    SharedPreferences.Editor editor = sharedPreferences.edit();
                    editor.putString( s: "companyName", currentProducer.getCompanyName());
                    editor.apply();

                    final Intent i = new Intent(context, ProducerProfile.class);
                    context.startActivity(i);
                } else {
                    final Intent i = new Intent(context, MapsActivity.class);
                    context.startActivity(i);
                }
            }
        });
    }
    // a default function
    @Override
```

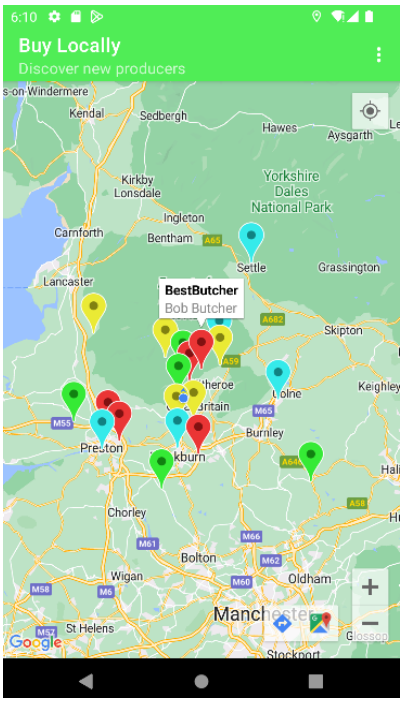
```
81    public int getItemCount() { return producersList.size(); }
84
85    // implementation of view holder class as it is very short
86    public class MyViewHolder extends RecyclerView.ViewHolder{
87        ImageView iv_typePicture;
88        TextView tv_companyName, tv_fullName, tv_description;
89        ConstraintLayout parentLayout;
90
91        public MyViewHolder(@NonNull View itemView) {
92            super(itemView);
93
94            iv_typePicture = itemView.findViewById(R.id.categoryIcon);
95            tv_companyName = itemView.findViewById(R.id.companyNameInList);
96            tv_fullName = itemView.findViewById(R.id.fullNameInList);
97            tv_description = itemView.findViewById(R.id.descriptionInList);
98            parentLayout = itemView.findViewById(R.id.oneLineListItem);
99
100       }
101   }
102 }
```

This should conclude all the main features, so I started testing the current version of the app.

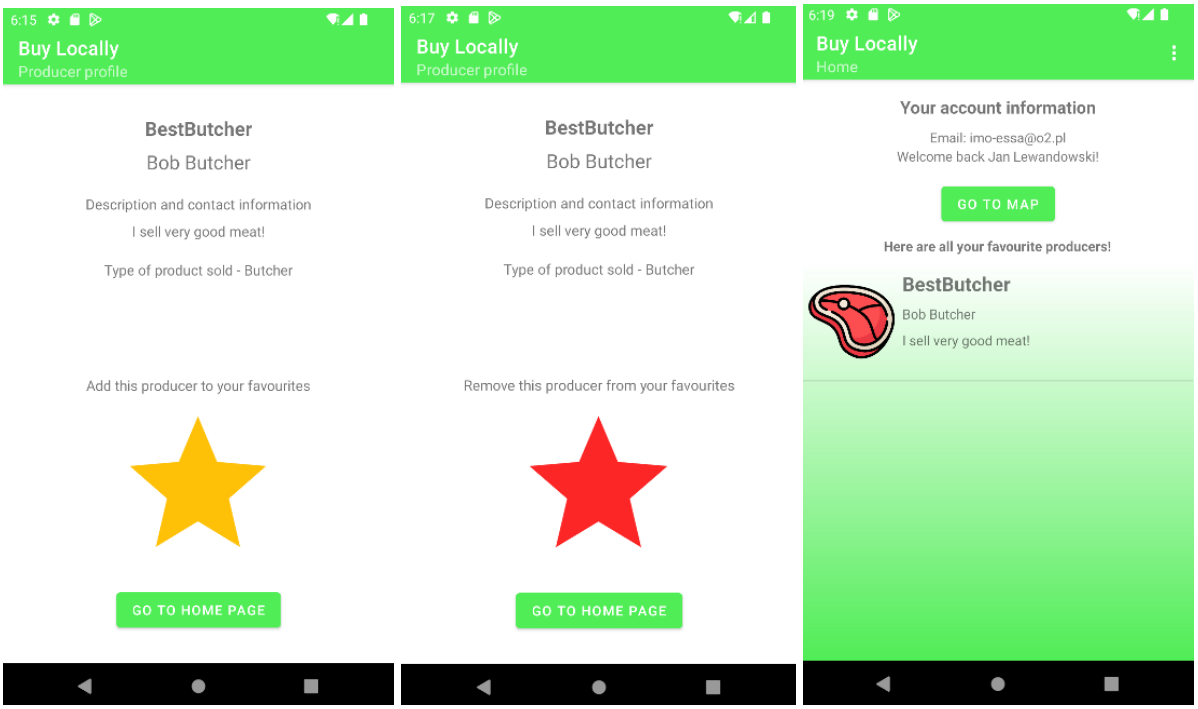
After first logging in, the list is empty, so I can only see one item, which is the empty item. If I click this empty item or the go-to map button, I get redirected to the map. The map opens on my current location, which is correct, I can see all producers which are in the database and different types have different coloured markers. When I click on a marker, the map centres on it and an info window pops up with the basic information—company name and the producer's full name.



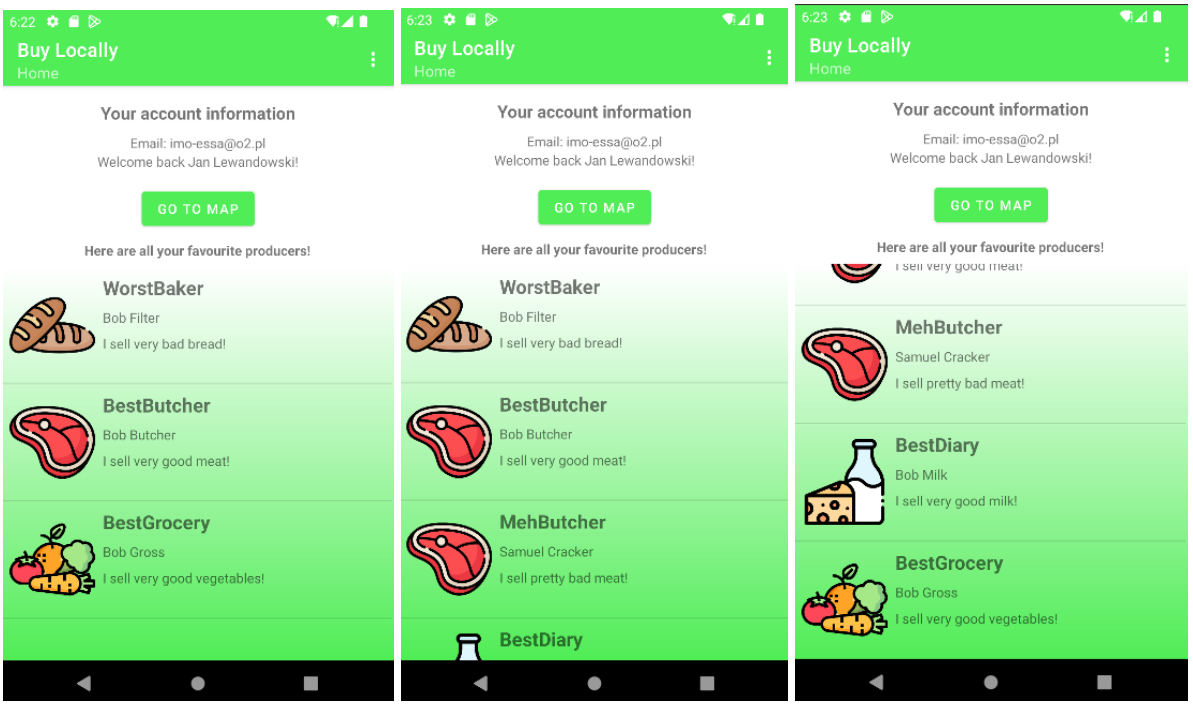




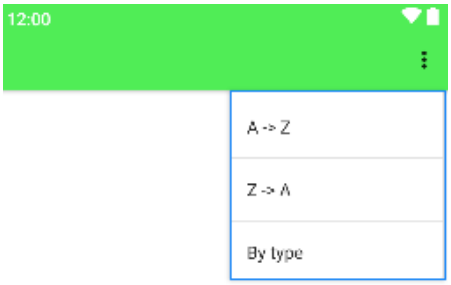
After clicking the info window, I get redirected to the producer profile activity and the correct info is displayed. After clicking the star, it changes colour to red and the text changes. When I click the go to home page button, I am sent to the home page and I can see this producer on my list, with the correct icon and other information.



After I add some more producers to my favourites, I can also see them on the list. If the list becomes too long to see it on the screen, I can scroll down to see the rest.



Then, also with some feedback from my stakeholders, I identified two more features that should be implemented—menus. One for the home page to sort the list and the second one for the map, to be able to choose which producer types should be displayed. This tutorial also helped me. Sorting functionality will make managing the list much easier, and to add this feature, I started with the home activity menu. First, I created a new layout file of this menu.



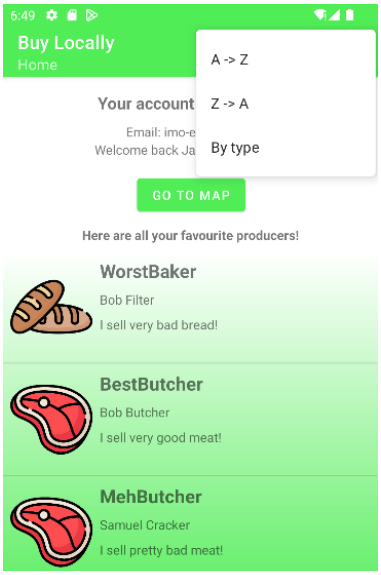
This menu allows the user to sort the list alphabetically, reversed alphabetically or by type. To be able to compare producers to each other, I needed to create comparators in the producer class.

```
    public static Comparator<ProducerModel> CompanyNameAZComparator = new Comparator<ProducerModel>() {
        @Override
        public int compare(ProducerModel p1, ProducerModel p2) {
            return p1.getCompanyName().compareTo(p2.getCompanyName());
        }
    };

    public static Comparator<ProducerModel> CompanyNameZAComparator = new Comparator<ProducerModel>() {
        @Override
        public int compare(ProducerModel p1, ProducerModel p2) {
            return p2.getCompanyName().compareTo(p1.getCompanyName());
        }
    };

    public static Comparator<ProducerModel> ProducerTypeComparator = new Comparator<ProducerModel>() {
        @Override
        public int compare(ProducerModel p1, ProducerModel p2) {
            return p1.getType().compareTo(p2.getType());
        }
    };
```

Then in the home activity, need to inflate the menu with my template and set up a click listener to sort the list when clicked in the correct order.

```
    // use the menu layout
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.sort_menu, menu);
        return true;
    }

    // when menu item clicked
    @Override
    public boolean onOptionsItemSelected(@NonNull MenuItem item) {
        switch(item.getItemId()){
            case R.id.menu_AtoZ:
                Collections.sort(allFavourites, ProducerModel.CompanyNameAZComparator);
                return true;
            case R.id.menu_ZtoA:
                Collections.sort(allFavourites, ProducerModel.CompanyNameZAComparator);
                return true;
            case R.id.menu_by_type:
                Collections.sort(allFavourites, ProducerModel.ProducerTypeComparator);
                return true;
        }

        return true;
    }
```

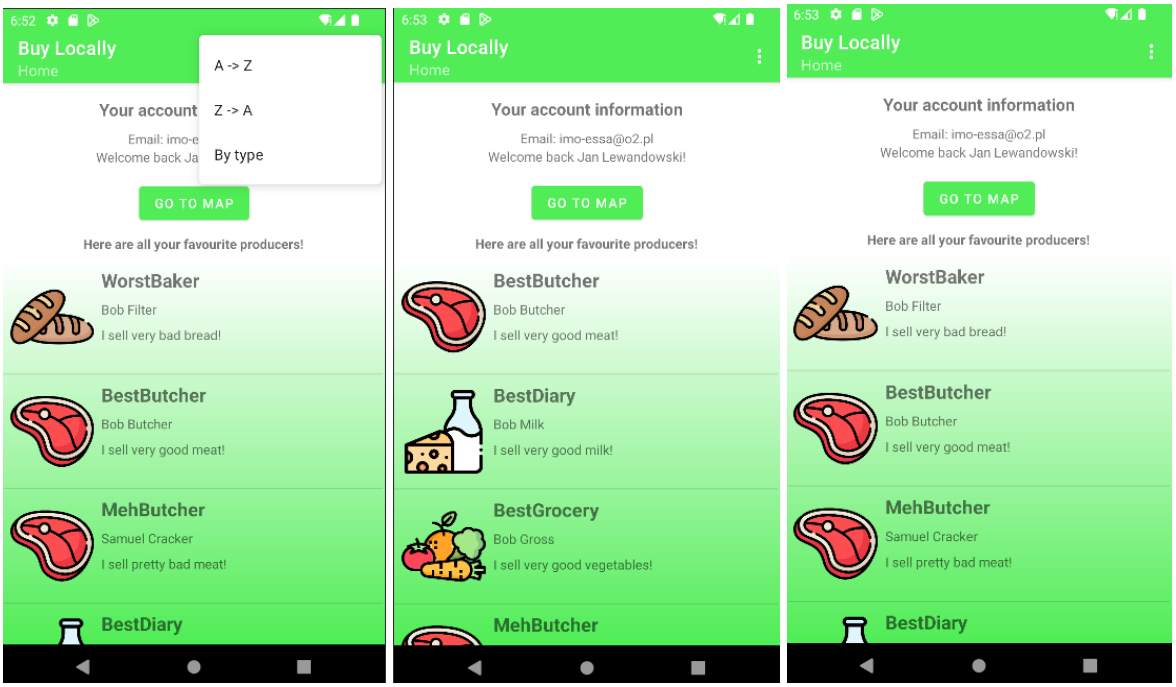
But when I click any of the options from the menu, the order of the list does not change. I found an error. When I click the menu options, the order should change, but it does not, so there must be an error somewhere. This error has to fixed, because if not then what would be the point of this menu if it doesn't affect the order of the list.



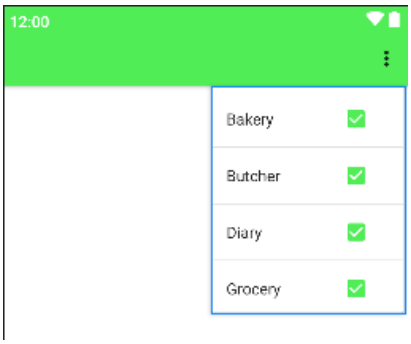
My error was that even though the list itself was being sorted, the list adapter did not know about it, so the order on the screen did not change at all. This fix of letting the list adapter know the order has been changed should resolve the issue.

```
120     @Override
121     public boolean onOptionsItemSelected(@NonNull MenuItem item) {
122         switch(item.getItemId()){
123             case R.id.menu_AtoZ:
124                 Collections.sort(allFavourites, ProducerModel.CompanyNameAZComparator);
125                 listAdapter.notifyDataSetChanged();
126                 return true;
127             case R.id.menu_ZtoA:
128                 Collections.sort(allFavourites, ProducerModel.CompanyNameZAComparator);
129                 listAdapter.notifyDataSetChanged();
130                 return true;
131             case R.id.menu_by_type:
132                 Collections.sort(allFavourites, ProducerModel.ProducerTypeComparator);
133                 listAdapter.notifyDataSetChanged();
134                 return true;
135         }
```

And now the menu works. When the option is clicked, the list is sorted alphabetically or by type. This is the correct behaviour and the error has been fixed.



The last feature to implement is the maps activity menu. I again started with the layout file for the menu.

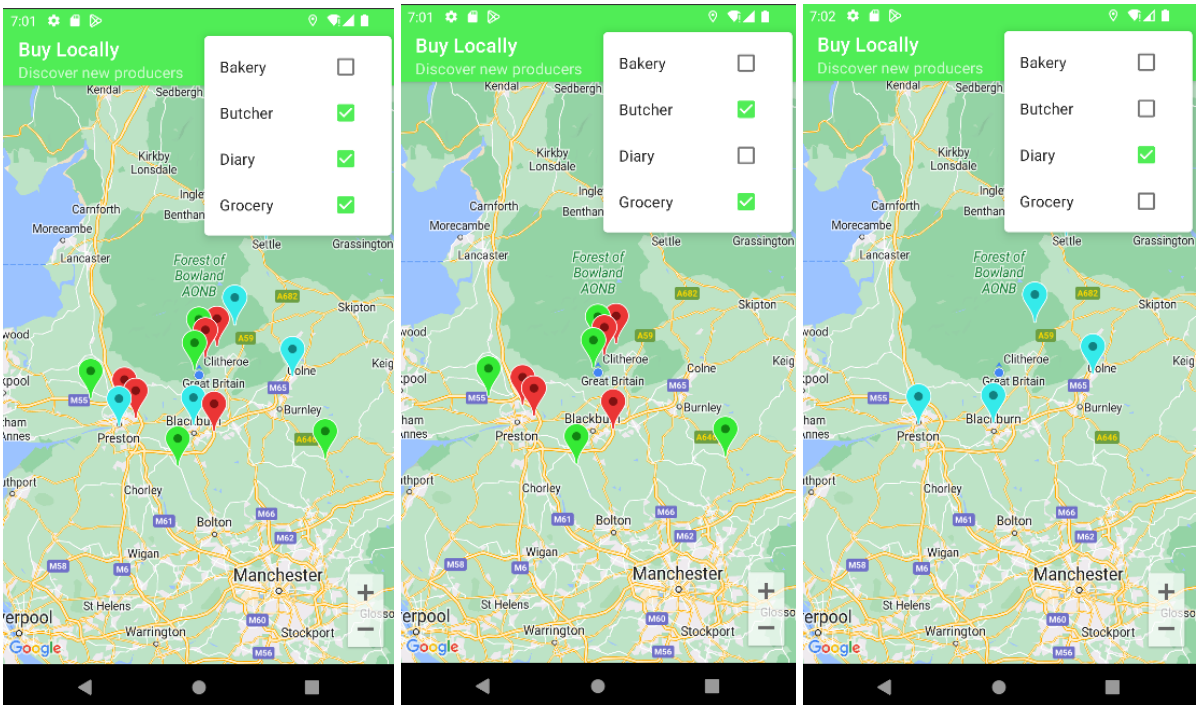


Then, I implemented this menu in the maps activity. The backbone is similar to the main page sort menu. After any change to the menu, I iterate over all markers and check if they should be currently displayed.

```
// to use the correct layout file
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    getMenuInflater().inflate(R.menu.map_menu, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(@NonNull MenuItem item) {
    int id = item.getItemId();
    boolean wasChecked = false;
    // checking and unchecking the options as needed
    if(item.isChecked()){
        wasChecked = true;
        item.setChecked(false);
    } else {
        item.setChecked(true);
    }
    String toChange = "";

    if(id == R.id.menuBakery) {
        toChange = "Bakery";
    } else if(id == R.id.menuButcher) {
        toChange = "Butcher";
    } else if(id == R.id.menuDiary) {
        toChange = "Diary";
    } else if(id == R.id.menuGrocery){
        toChange = "Grocery";
    }
```

```

    if(wasChecked){
        chosenTypes.remove(toChange);
    } else {
        chosenTypes.add(toChange);
    }
    // amending the list of producers shown
    DBHelper databaseHelper = new DBHelper( context: MapsActivity.this);
    for(Marker marker : markersList){
        marker.setVisible(false);
        String name = marker.getTitle();
        ProducerModel producer = databaseHelper.getProducerByNameOrID(name,  ID: -1).get(0);
        if(chosenTypes.contains(producer.getType())){
            marker.setVisible(true);
        }
    }

    return true;
}
```

Then I tested this menu, and it worked perfectly, at first all producers are displayed, but then when I click the options, all producers of a given type are hidden or shown, according to the menu options.



Final testing table

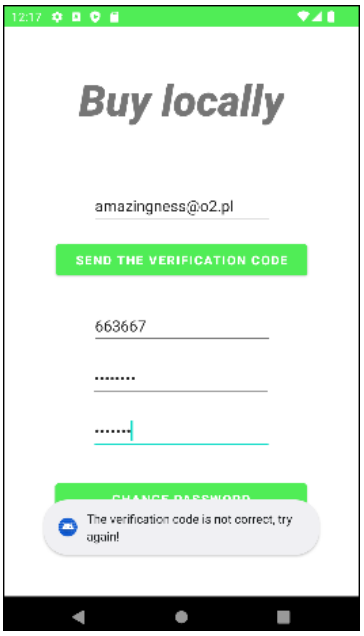
Test number	Process tested	Input	Type of input	Expected output	Actual output
1	Info window on the map	Once clicked	Normal	Opening the producer profile page with the correct details	Success
2	Empty list element	Once clicked	Normal	Opening the map to add producers, because the list is empty	Success
3	Star in the producer profile	Once clicked	Normal	Adding the producer to favourites and displaying them later on the list on the main page	Success
4	List element on the main page	Once clicked	Normal	Opening the producer profile page with the correct details	Success
5	Sort options	Once clicked	Normal	Sort the favourites list accordingly	Success

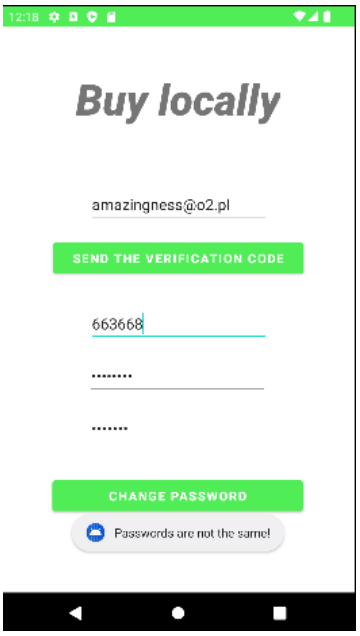
Evaluation

Testing for robustness

In order to test for robustness, I made use of beta testing by giving the application to a potential end user and asking them to enter data incorrectly and ignore instructions in order to try to break the program. No errors, when incorrect data is inputted or when the user clicks on incorrect buttons or outside of them, the correct error message is displayed. Here are examples of correctly displayed error messages.

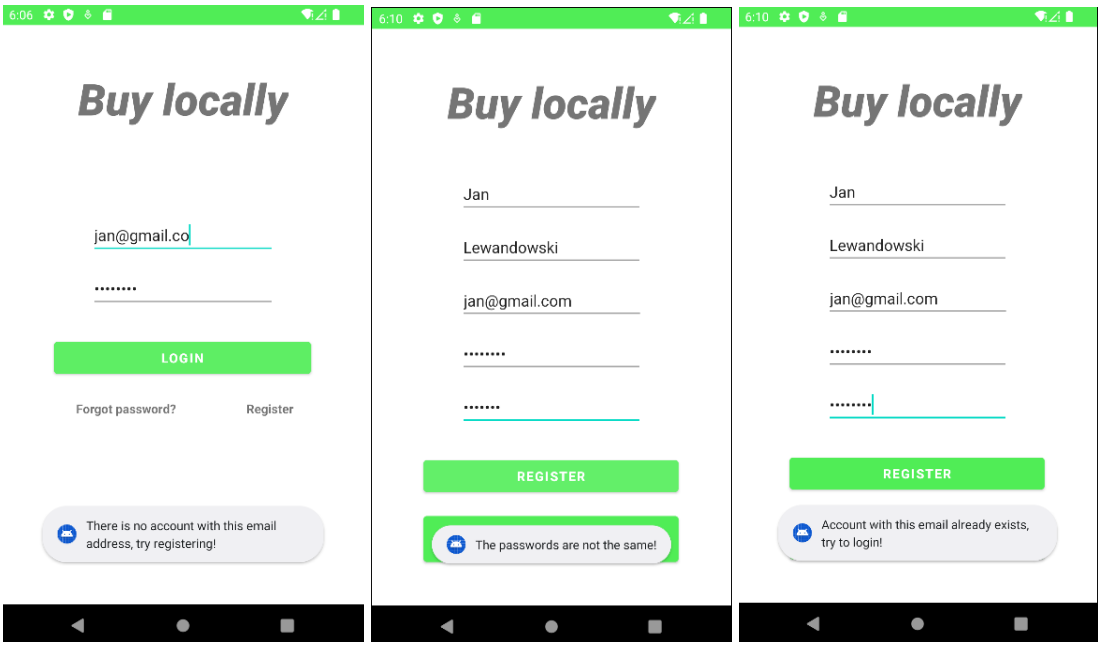
In the forgotten password activity, when the user inputted the wrong verification code or when password and confirm passwords fields are not the same, a corresponding error message is displayed. In this case, the verification code is 663668, so to test robustness I inputted an incorrect code, 663667. The password has not been changed, and a message popped up - “The verification code is not correct, try again!” - which is the expected behaviour. Similarly, when I put in the correct code - 663668, but the new passwords are not the same, the first one is 1234567A, while the second one is 1234567, an error message popped up - “Passwords are not the same!”, which is the correct behaviour.





Next situation, in the login activity and register activity, if there is no account with the given email address, or password and confirm password fields are not the same, or when the user tries to create an account on an email that is already used for another account, the correct corresponding error message is displayed. When I put in an incorrect email address that’s not in the database, such as jan@gmail.co, an error message pops up saying “There is no account with this email address, try registering!”, which is the correct behaviour. Then, when I try to register a new account and the password and confirm password fields are not the

same, for example 1234567A and 1234567, the error message “The passwords are not the same!” is displayed, which is the right behaviour. When I try to register an account with all correct details inputted, but an account with the given email address already exists, like in this case, an account for jan@gmail.com is already in the database, an error message “Account with this email already exists, try to login!” is displayed, which is the correct behaviour.



The program passed all robustness tests successfully, displaying the right, meaningful error message to inform the user that some of the details given are erroneous.

Testing for usability

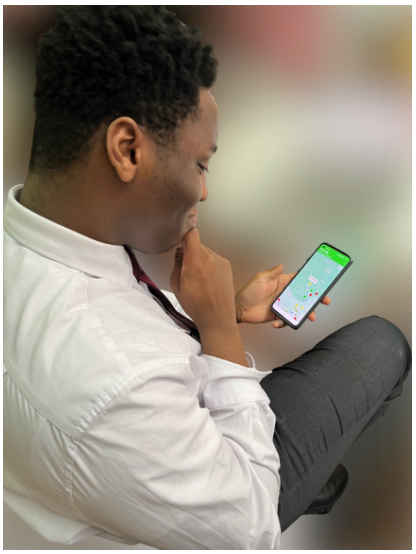
Below are the questions I asked my stakeholders, Daniel Uko and Szymon Lis.

Question	Justification
Did you have trouble navigating the app's interface or finding the information you were looking for?	The user experience is crucial to the success of my app. By asking this question, I can identify any areas where the app's interface or organisation may need improvement. If users have difficulty navigating the app or finding the information they require, they may become frustrated and give up on the app. By addressing these issues, I can improve the overall user experience and increase engagement with my app.

Did you encounter any errors or bugs while using the app?	Technical issues can be a major source of frustration for users. By asking this question, I can identify any errors or bugs that users may encounter while using the app. I can use this information to improve the app's stability and ensure that users have a seamless experience while using the app.
Was it easy to create an account and log in to the app?	The account creation and authentication process is a critical aspect of the app's usability. The authentication process has to be as straightforward as possible. By asking this question, I can identify any issues with the account creation and authentication process and make improvements to ensure that users can easily create an account and log in to the app.
Are you able to add the producers to your favourites and see the list on the main page?	This question is important to test the ease and effectiveness of the app's favourites feature. The users should not have any difficulty adding producers to their favourites or finding their list of favourites on the main page. This feature can help users keep track of producers they are interested in and make it easier for them to find and purchase products from those producers in the future.
Was it easy to view and navigate the map, and were the producer locations clearly marked and easy to find?	The usability of the app's map feature is critical to the success of the app. If users have difficulty viewing or navigating the map, or if the producer locations are not clearly marked or easy to find, they may become frustrated. By asking this question, I can identify any issues with the app's map feature and make improvements to ensure that users can easily view and navigate the map.

Do you like the feel of the application? What could be improved or more intuitive?	This last question is a broader one, here the end-users can express their feelings with the use of my application and comment on any features they don't like or which should be added or made more intuitive.

Below are the answers I received from Daniel and Szymon.



1. Did you have trouble navigating the app's interface or finding the information you were looking for?

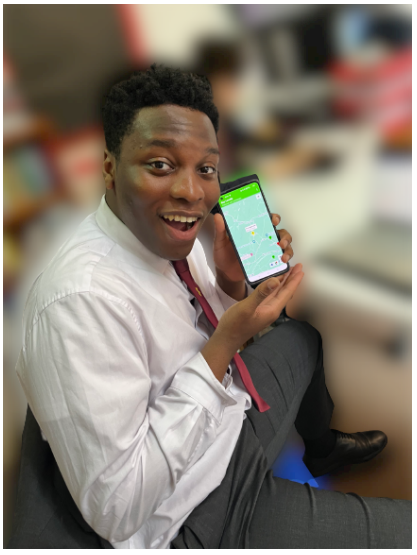
"All the buttons are easy to recognize and are labelled according to what it does. The UI is really intuitive, and I have no problems using it. However, I think the green colour used might be a little too bright for some people."

"I think the app navigation is all right, all buttons are helpful, and they do what they are meant to do. I think when the app gets bigger with more functionalities, it might be difficult to connect everything together only using buttons. In my opinion, a sidebar would make the navigation even better."

2. Did you encounter any errors or bugs while using the app?

"All the features that are implemented work correctly, and I did not encounter any bugs or errors. All the buttons link to correct pages and the database works as well."

"In terms of robustness, I have no comments to make. I did not encounter any bugs or crashes or overly long loading times. "



3. Was it easy to create an account and log in to the app?

"Yes, it was very easy to create an account and log in to the app. The process was straightforward and intuitive, and I didn't encounter any issues or confusion. I appreciated that I could use my email address to create an account, as I prefer not to link my social media accounts to apps. Overall, I was impressed with how easy it was to get started with the app."

"I found it quite easy to create an account and log in to the app. The process was fairly straightforward, however I think it would be useful to add a different method to sign up, especially using social media accounts."

4. Are you able to add the producers to your favourites and see the list on the main page?

“I see all the producers on the map with their correct coloured markers. When I click the info window, I am redirected to the producers page and I can add them to my favourites. This function works as I can then see the whole list of favourites on the main page.”

“Yes, I found it very easy to add producers to my favourites list and see the list on the main page. The process was intuitive, and I appreciated that I could easily access my favourites from the main page. All producers on the map are distinctively marked, so I could recognize them and them to my favourites after clicking the marker and going to their profile.”

5. Was it easy to view and navigate the map, and were the producer locations clearly marked and easy to find?

“I enjoyed using the map, but I think the icons could be smaller, as sometimes they block a considerable part of the screen.

“Yes, it was really easy to navigate the map, as I am used to the look of Google Maps. I can differentiate between producers on the map and easily see them as the icons are very distinctive.”

6. Do you like the feel of the application? What could be improved or more intuitive?

“My impression of using the app was very positive. In my opinion, the application would be improved with the addition of a product system with a cart inside the app, and easier communication with the producers. The chatting system would allow me to communicate with producers directly.”

“I really liked the application and I only have a few small changes which I think could improve it. The addition of some kind of product system, which would allow me to see the products offered by each producer and add them to a cart. Also, I would appreciate the addition of a chatting system to directly contact producers and the producer’s profile page should contain a link to their website.”

Success of the solution

Success criteria	Comment	Completed
Search box		
Search by the postcode	I have not created a search box for the map because it would require me to rebuild the whole map activity	Not met
Search by producer	I did not create the search box for finding producers, but I created an option to only show producers of a given type on the map, which	Partially met

	essentially allows the user to search by producer type. I implemented and tested this feature in prototype six	
Search by products	I did not implement any of the products' related features, due to limitations of a SQLite database and not enough time of development.	Not met
Map		
Meaningful icons	The icons are very descriptive, the colours are different from each other, and they are displayed correctly. I implemented this feature in prototype four, there are four different colours of markers for the different categories - Butchers, Bakeries, Diaries and Groceries.	Fully met
A box to put the postcode	I implemented an even better function, with a click the user is located, which is much easier than having to input the address manually. Already in prototype three I was able to locate myself on the map using the button in the top right corner.	Fully met
Using conventional colours	The main colour matches between activities and the maps activity has an exact, very well known Google Map. I added the Google maps activity in prototype three and it is working since then.	Fully met
Log in screen		
Register button	There is a working button to register a user, which was implemented in prototype one.	Fully met
"Register as a producer" button	There is a working button to register a company as a producer. I implemented this feature in prototype four as well as the producer profile page.	Fully met
Password box	There is a password box working correctly. This feature was one of the first ever implemented in this project in prototype one and has been working since then.	Fully met
Database of customers		
Login details fields in the database	Database works, all users are stored with their details	Fully met

Storing user favourites	Created a relational database with working link layer	Fully met
Storing information about all the producers	Created a table for all producers, working and storing all details	Fully met
Chatting system within the app		
A “start chatting” button in producer page	Chat features not implemented due to limitations of SQLite and underestimation of the complexity of this feature.	Not met
Chatting page itself	Chat features not implemented due to limitations of SQLite and underestimation of the complexity of this feature.	Not met
Notifying the user when they get a new message	Chat features not implemented due to limitations of SQLite and underestimation of the complexity of this feature.	Not met

Usability features

Feature	Met / Not met
Login page	The login page allows users to log in after they register the account. It also contains a button to log in as a producer. It is fully working and tested, it displays the correct error message according to the details given.
Register pages	The register page allows a new user to register their account and store it in the database, so that they can log in and use the app. It is fully working and tested, it also displays the correct error message if the details are wrong.
Main page	The main page shows the details of the current account logged in, and a list of all producers the user added to favourites. After a producer on the list is clicked, the user is redirected to the correct producer profile. The user can also sort the list.
Map activity	Map activity displays all the producers on the map, with different markers colours for different types of products. The user can also select which

	type they want to be shown. The user can navigate the map, zoom in, zoom out and when the producer's icon is clicked, a window pops up, which takes the user to the producer's profile page.
Producer profile	The producer profile page displays all the information about the given producer, and allows the user to add their company to favourites. In future development, this page would also display products of a given producer.
Chatting system	Not met, I didn't have enough time to implement the chatting system and I also underestimated the difficulty to create one, as it would require a lot more programming. Also, the database would have to be hosted externally for the chatting system to work.

In conclusion, I managed to meet almost all the success criteria. I overlooked the difficulty and the time necessary to complete some features, such as a chatting system, which could essentially be an app on its own. Using SQLite also did not allow me to do that, because a database stored locally on the user's device will not suffice. I created the producers' profile page, but I should also add the option for producers to put their website links there, as highlighted by one of the stakeholders in the questionnaire. I did not manage to implement a product system and a cart inside the app. This feature is wanted, and would make using the app much more convenient. I overlooked the complexity of storing many products from different categories and assigned to many producers. It would take much more time to plan and modify the database and create a ListView in a modified producer profile.

Maintenance and development

Maintenance

The application was created using modular programming techniques in an object-oriented language – Java. This, and comments throughout the whole code, makes the program easy to maintain. The comments let me know what each component does, which is very helpful when I want to change something in the code, especially that my application has a dozen different Java classes, and it would be very difficult to navigate without comments. The modular structure of object-oriented programming will make further development much easier and more efficient. The visual structure of the Android Studio, with all files ordered tidily, which clearly shows the structure of the program and allows me to swiftly edit and expand in the future development.

Further development

The tables below outline how any partially or unmet success criteria and usability features might be addressed in the future, as well as the limitations of the project and plans for further development.

Success criteria	Further development plans
Chatting system	To implement a chatting system, I would have to use a real-time database, such as Firebase. SQLite does not allow me to create a chatting system. If I were to implement a chatting system, I would use a different database system, for example using Google's Firebase, which makes it really easy to create and maintain a real time database. This would require reworking the database code to adapt it.
Products system	To implement the products' system, I would create a new table and then modify almost all producer pages to include products. I would also have to modify the producers' database. The products would be displayed in the producer profile activity. Also, the producers would have a new activity for adding the products.

Limitation	Further development plans
Application is only available on smartphones with Android	To reach more people and allow a wider range of people to use my application, I would create a port of the application for iPhones, and even a website to use it on a desktop computer. To overcome this limitation, I could create the app one more time in the iOS environment and create a website so that the desktop is also supported, which would be quite tedious. If I wanted my app to be available on all platforms, I would probably re-implement it from the ground up, for example in Flutter, which supports all platforms from a single codebase.
I am a single developer	As mentioned before in the design stage, I am a single developer and I am not working on this project full-time, so I wouldn't be able to implement every single feature that might be wanted. To overcome this limitation in the future, I would need to assemble a team of developers who want to work on my project, which would be costly.
Time	As mentioned before in the design stage, this project is

	time-bound, so I haven't managed to implement all the features that I planned to implement, such as the chatting system, which would take a lot of time to create. To overcome this limitation, I would have to work on the project full-time, dedicating most of my time to it.
Experience	This was my first project in Java and Android Studio. I have learnt a ton along the way, and starting again, I would do some things differently. If I develop this app further in the future, I would already have a base of experience, which will make the development much more efficient and effective. Overcoming this limitation will probably take the longest as I can only gain experience by building the application, which is a gradual process.
Offline database	Due to using SQLite as the main database system, the data is only stored locally on the user's device. If the app was to be released for the broader public, the database would have to be ported into external hosting, such as Firebase by Google, which I think is perfect for this app, especially in its early development stage. This would allow the app to fetch data - locations, descriptions, details - from a global database which is seen by all users in the same way. The current app is not really using the client-server model, which is necessary to overcome this limitation.

Bibliography

https://github.com/Musfick/AndroidDevelopmentKit/blob/master/JavaMailAPI.java

https://github.com/trycourier/courier-java

https://developers.google.com/maps/documentation/android-sdk/start

https://github.com/delight-im/Android-SimpleLocation/blob/master/Source/library/src/main/java/im/delight/android/location/SimpleLocation.java

https://developers.google.com/maps/documentation/android-sdk/location#my-location

https://stackoverflow.com/questions/32083913/android-gps-requires-access-fine-location-error-even-though-my-manifest-file

Code