

Android 2 Project - Group 5



Anastas Kuyumdzhev, Lyudmil Pashayanov, and Ivan Jirov

INTRODUCTION	3
MoSCoW	4
GUI	5
Server used	6
Login functionality	7
User map handling functionality	8
Database structure	9
Notification implementation	10
Interesting implementation details	10
Tasks by group member	10
Anastas Kuyumdzhev	10
Lyudmil Pashayanov	11
Ivan Jirov	11

Extra functionality	11
Libraries:	11
Extra project functionality:	11
Bugs and unexpected problems during development	13

INTRODUCTION

This document will provide important details about the development of our application for the course ANDR2. This application will take the basics of the proposed one and also extend them with some additional functionality.

Users need to have accounts to access the application. You will be able to make an account with your email or by linking your Google account. You will then be able to edit your profile, changing your photo, username and phone number.

As the main idea of the proposal is to have an application that tracks different people in your neighbourhood, we will use that idea and enhance it by adding the ability to post offers for items you might want to sell. A user will be able to post offers to their product page and other users will be able to indicate interest in those offers. When that happens, the one that posted the offer will receive a notification and be able to contact the other person by phone number.

Users will be visible on the map based on a radius option, as some people might want to look for offers outside of just their neighbourhood. Tapping on a user will show you one of their products, their username and their phone number. If you are interested in more of their items, you can also go to a list of all of them.

A list of all the products without a map will be available as well.

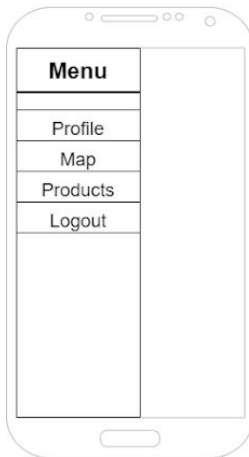
MoSCoW

MoSCoW Functional Requirements				
Features	Must have	Should have	Could have	Won't have
Users can sign-up and subsequently log in with email and password	◆			
Users can sign-up and subsequently log in via Google using Gmail account	◆			
Users can see their and other people's location on the map	◆			
Users can add items they wish to sell	◆			
Users can check what other people have on sale at any time	◆			
The app will allow users to add friends				◆
Users can navigate to other people's list of items by clicking on their profile from the map		◆		
The application will allow for profile management	◆			
Profile information such as email, name and phone will be visible to other users	◆			
Users can specify an area for which they can receive notifications if a new item is up for sale			◆	
User manual				◆
Application documentation	◆			
Users can send out push notifications to others in the area when they add a new item			◆	

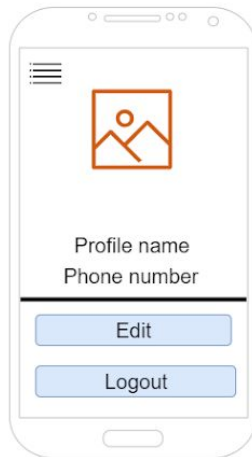
MoSCoW Non-Functional Requirements				
Features	Must have	Should have	Could have	Won't have
App displays relevant error messages	◆			
App asks for user permission	◆			
Final app does not crash during workflow	◆			
Application provides hints for new users				◆
Application is built using colors that are suitable for color blind people			◆	

GUI

Menu options



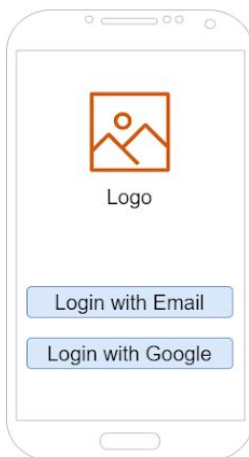
Profile



Profile (Edit)



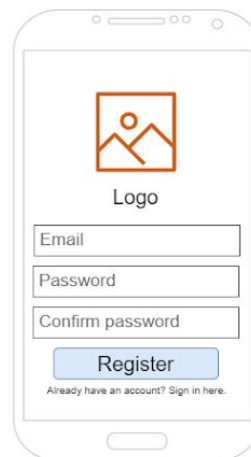
Login (Default)



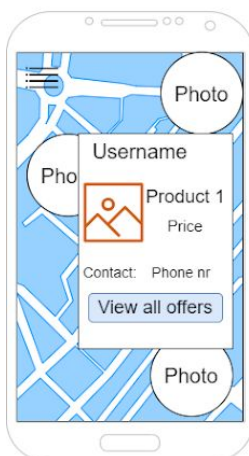
Login (Email)



Register



Map



Products



Product Offer
(Post and Edit)



Server used

For our project, we chose to implement Firebase Firestore. Reasons why we chose this database option are:

We used this Stackoverflow answer to better determine whether to use Firestore or the Realtime Database both offered by Firebase:

<https://stackoverflow.com/questions/46549766/whats-the-difference-between-cloud-firestore-and-the-firebase-realtime-database>)

- **Better querying:** In the Realtime database, We can only sort or filter on a property in a single query, not both sort and filter on a property while In the Cloud Firestore, You can chain filters and combine filtering and sorting on a property in a single query.

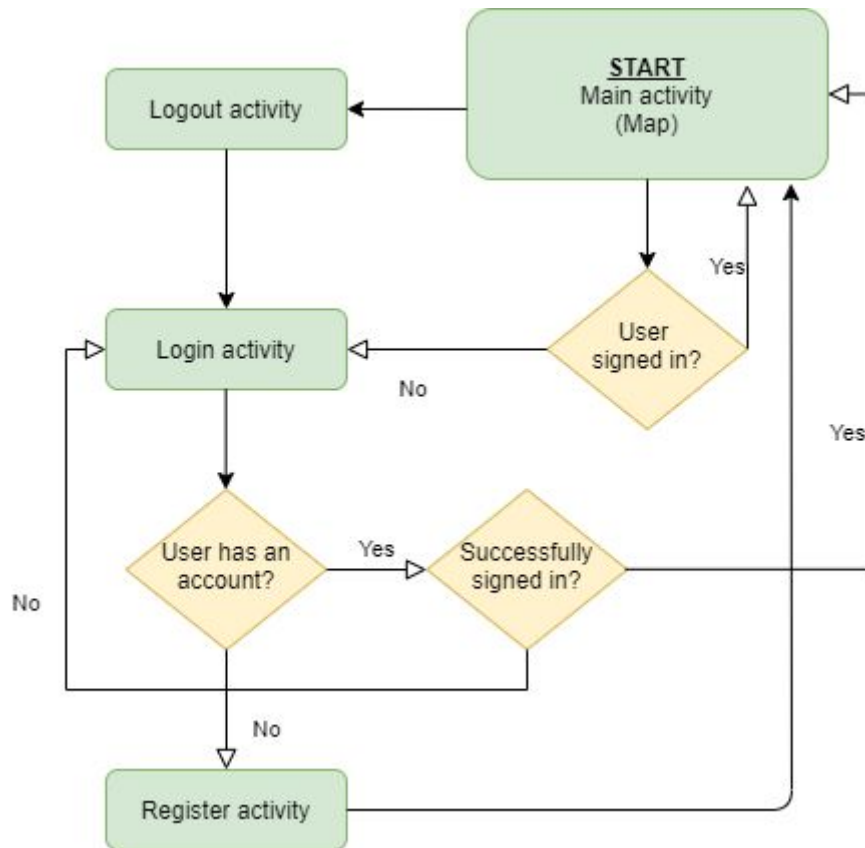
If you want to fetch data in descending order then Cloud Firestore is very useful for you but for the Realtime database, there is no query feature available.

```
citiesRef.orderBy("name", Direction.DESENDING).limit(3);
```

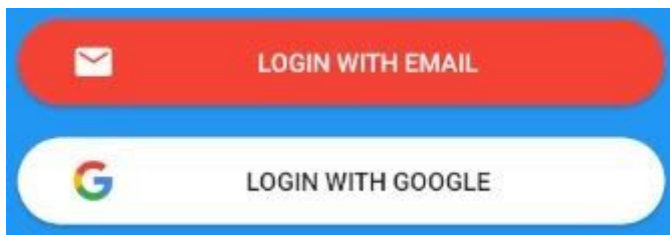
- **Security:** In the Realtime Database, We need to validate data separately using the validate rule but in the Cloud FireStore, data validation happens automatically.

And overall Firestore is a newer product that is recommended for new applications, as is our case.

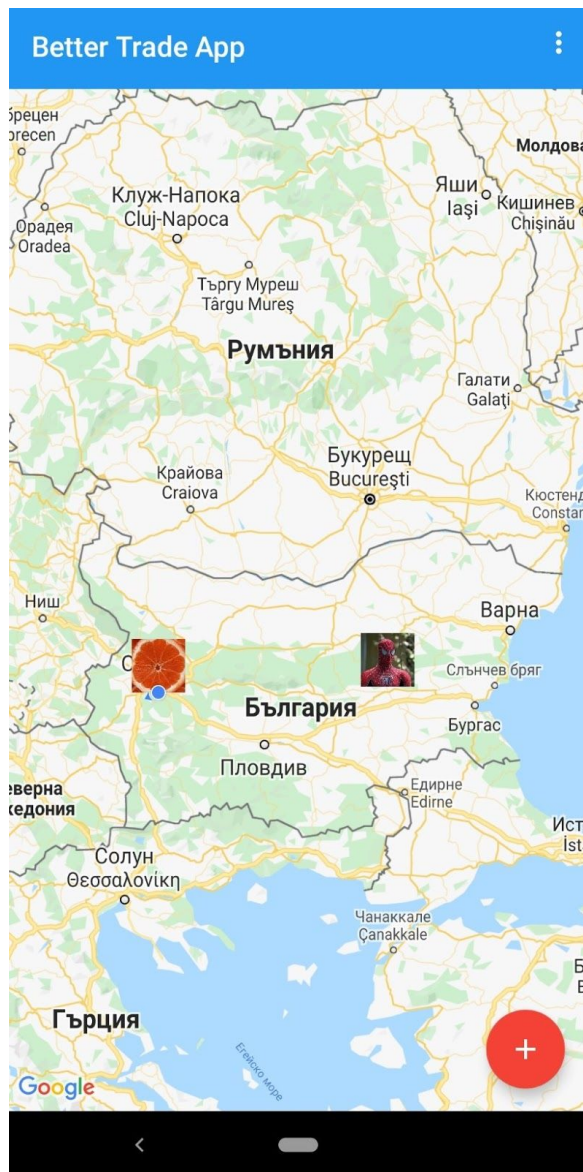
Login functionality



For the login we offer two options - Google and email + password:



User map handling functionality

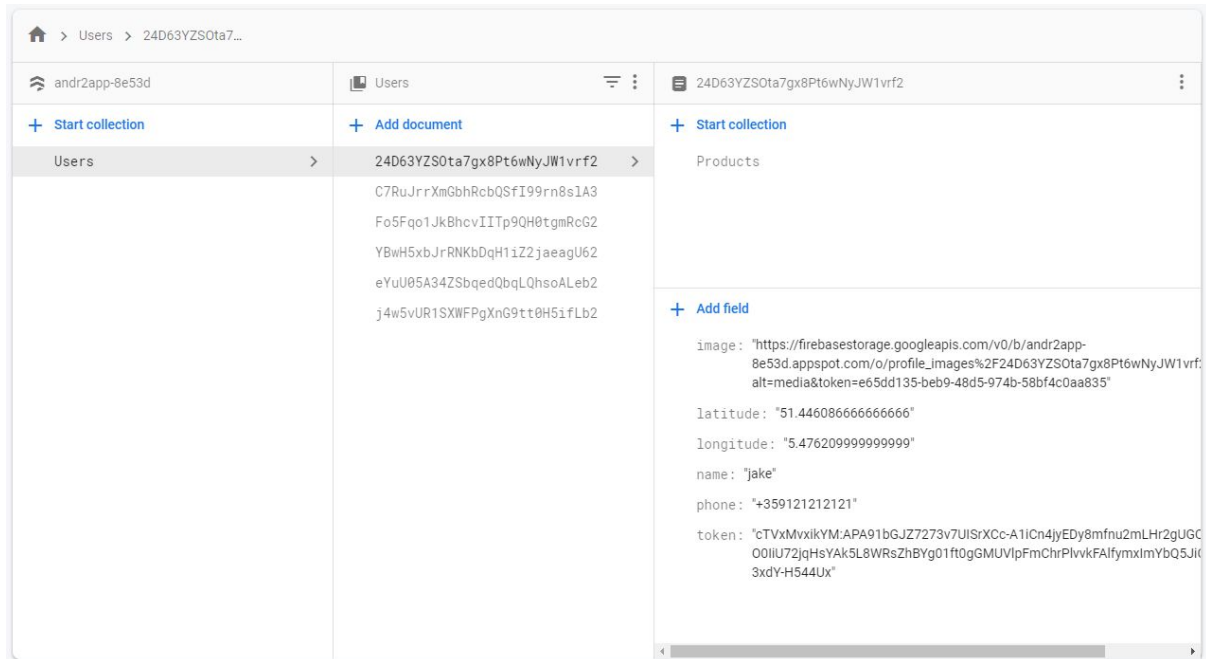


Whenever a user enters the app for the first time, we send him to the register page. When an account is created the user is sent to the main page, where all users will be sent, when they have logged in with their account.

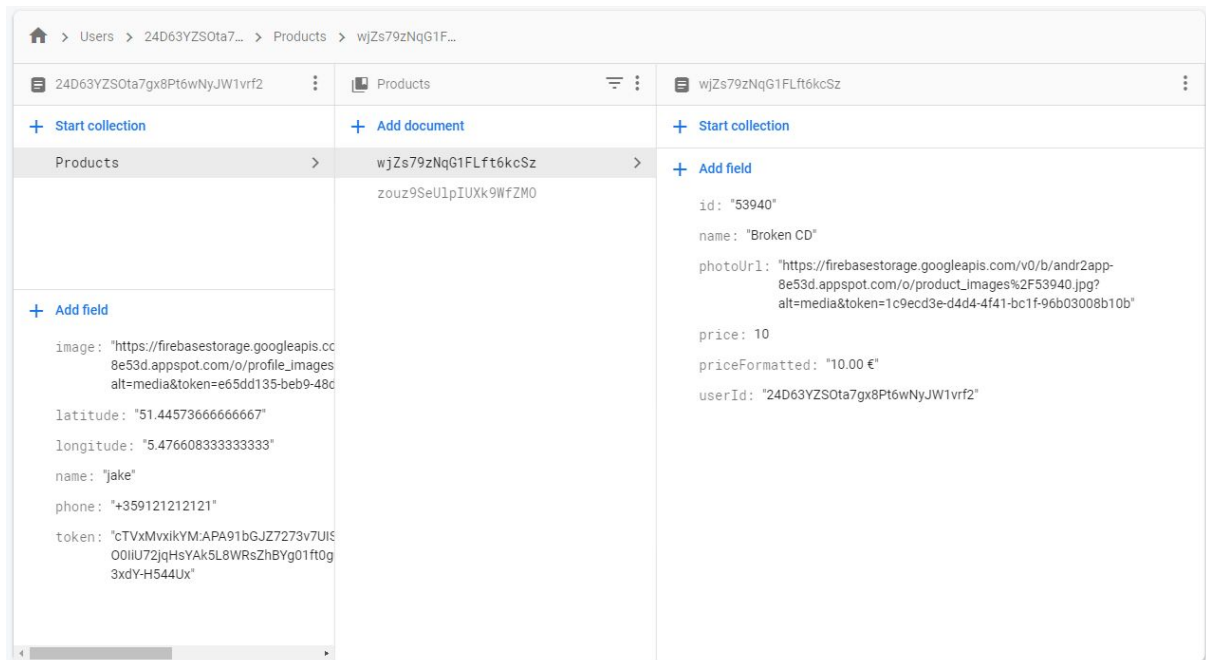
If the user doesn't have his location turned on, our application will prompt the user to turn the service on. Whenever the application detects that the location service is on, the google maps will zoom on the user's location.

From that point on, the user can explore what is sold around him. Every avatar visible on the map, represents a registered user, who may have items for sale. Whenever you see that an avatar is moving, that means that the current user is logged in the app, and you see his/hers real time location.

Database structure








We have documents who are different users and they are grouped in one collection - Users. Each user has a profile image, name, and phone fields. There are also latitude and longitude fields which determine the user's location on the map.



Furthermore, each user has a Products collection, with name, price and image fields. A user can have multiple products for sale.

The images point to a storage link:

gs://andr2app-8e53d.appspot.com > profile_images				Upload file		
<input type="checkbox"/>	Name	Size	Type	Last modified		
<input type="checkbox"/>	 VXkwu1HWr3TjV82XH5Wf1F2aglo1.jpg	12.36 KB	image/jpeg	May 6, 2020		
<input type="checkbox"/>	 nGt3NwK48Fgd0tkkbtn7ti5A6U63.jpg	1.64 KB	image/jpeg	May 7, 2020		

gs://andr2app-8e53d.appspot.com > product_images				Upload file		
<input type="checkbox"/>	Name	Size	Type	Last modified		
<input type="checkbox"/>	 168062.jpg	1003.69 KB	image/jpeg	Jun 20, 2020		
<input type="checkbox"/>	 253378.jpg	962.37 KB	image/jpeg	Jun 22, 2020		
<input type="checkbox"/>	 332648.jpg	1.36 MB	image/jpeg	Jun 12, 2020		

The Storage consists of the profile and product images. The profile picture names are constructed by appending “.jpg” to the individual user’s UID.

Notification implementation

The implementation of notifications is applied when a person is looking at an individual product. If that user wants to indicate interest in that product, they can press the button, which will send a notification to the product’s poster.

Notifications are sent using the Firebase Cloud Messaging. Each user has a unique token that is updated regularly. This token is used to specify the person that the notification is sent to. I set up a small API that makes a post request using a `FirebaseService`. The notification is built in that service and sent to the specified user.

Interesting implementation details

About notifications - The fact that older versions of Android do not need notification channels is interesting to me and was sort of a barrier to making things work well for everyone, at least until I realized it. I can see channels as something useful in a bigger app and could definitely make the implementation of different types of notifications more independent and modular, while keeping it fairly easy. As with all things, once you have learned how to do it once, the concepts behind it become much clearer and future implementations can become significantly better.

Tasks by group member

Anastas Kuyumdzhev

- App GUI layout
- Implementation of product functions:
 - Product CRUD
 - Product views (all products, your products, individual product)
 - Notification of interest in product
- Unit tests

Lyudmil Pashayanov

- Implementation of Google maps
- Setting up real time location update on all users
- Setting up user's avatars appear on the map
- Setting up steps tracking sensor

Ivan Jirov

- Project setup - Git, Firebase (Authentication, Firestore, Storage, Test Lab)
- Authentication - Register and Login via Google or email and password
- Profile management functionality
- Library integration
- Navigation
- Instrumentation tests using Espresso

Extra functionality

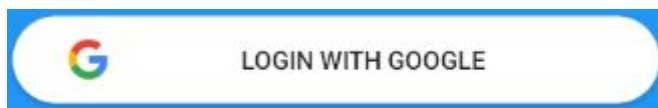
Libraries:

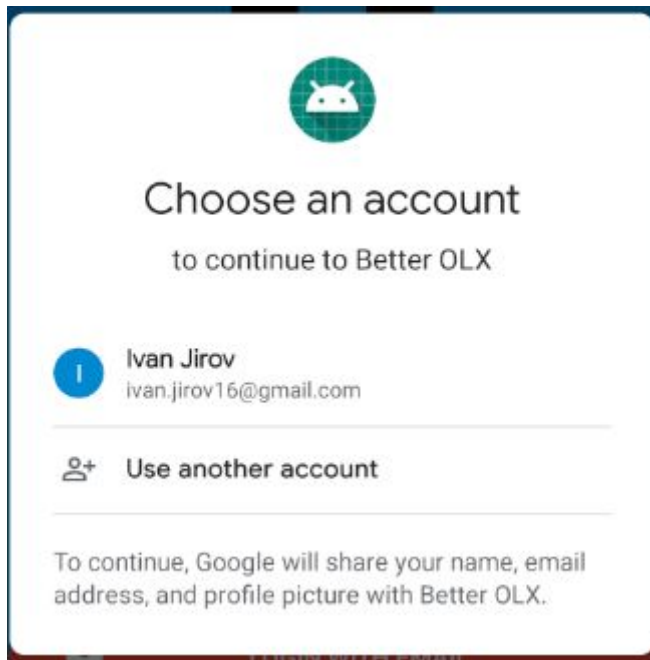
- Image cropper: <https://github.com/ArthurHub/Android-Image-Cropper>.
We used this image cropper library so that the user can exactly proportion their profile images. This is extra functionality that we implemented in order to improve user experience for our application.
- Glide for image loading from an url: <https://bumptech.github.io/glide/>. An alternative is Picasso.
Glide is very simple to use and loading images from an url can be done like this:

```
Glide.with(fragment/activity)
    .load(url)
    .into(imageView);
```
- Image view library: <https://github.com/hdodenhof/CircleImageView>.
We used this library to display profile and product images in a square image view.
- Country code picker: <https://github.com/hbb20/CountryCodePickerProject>.
We used this library to display different country codes from which the user can choose from while entering his phone number.

Extra project functionality:

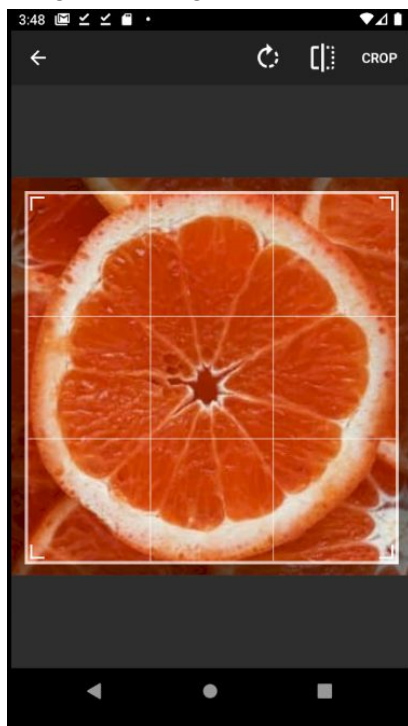
- Google sign in:





We implemented Google sign in into our project as we feel like it offers a streamlined signup experience to the user.

- Image cropping:



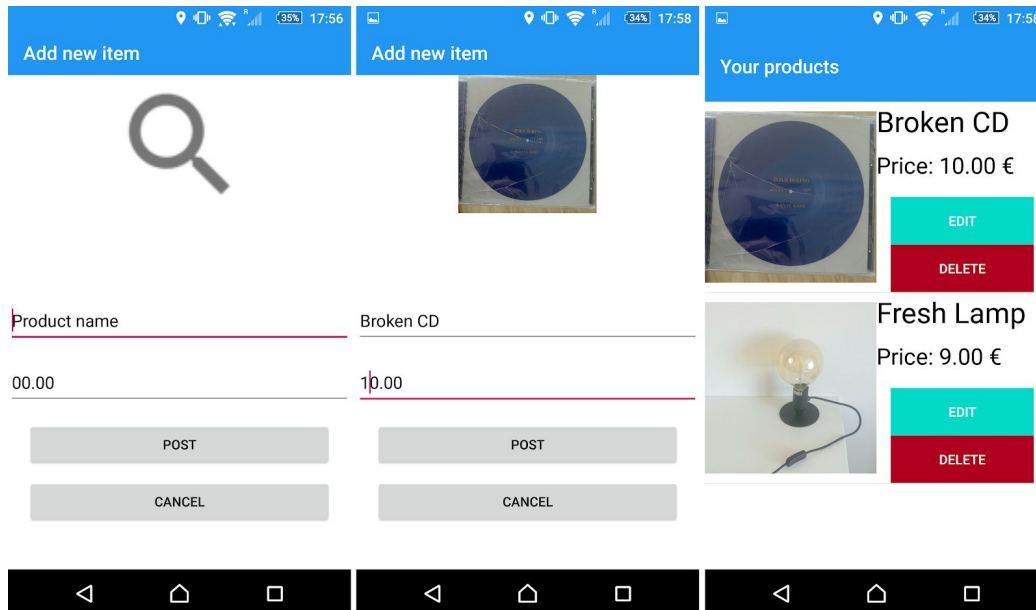
By offering image cropping functionality we improve the user experience by providing the users with a way to customize their profile pictures.

- Country phone code picker:

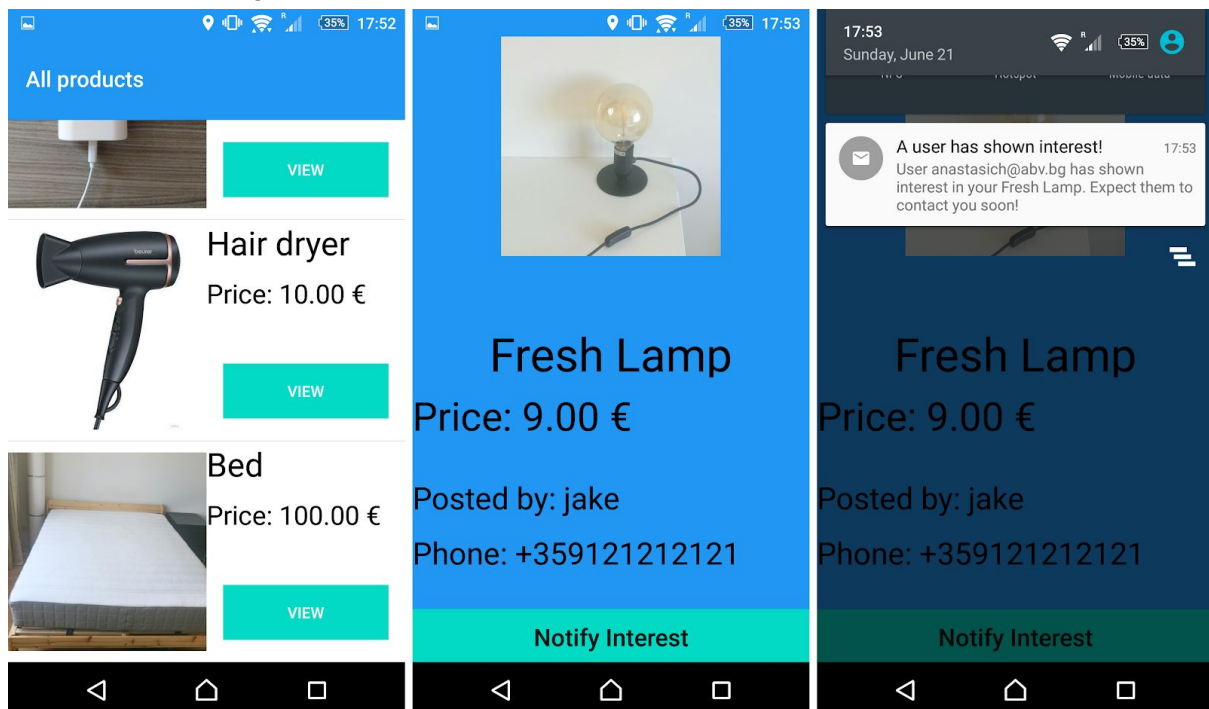


Using this functionality, the user can select the country code which is valid for their country.

- Product functions:



Adding a product (Deletion happens when tapping the button, editing takes you to a screen similar to the adding one)



Viewing all items, an individual item and signifying interest in an item by sending a notification.

Bugs and unexpected problems during development

- During development of the real-time location update of the users, a bug appeared where the users appeared to be in Africa and not on their real location. In the end it

turned up that the latitude and longitude values for the map were swapped when saved in the Database and therefore the locations were wrong.

- Some Espresso tests were failing due to supposedly missing elements in the generated view. This was due to the quick test execution and the views needed more time to be loaded. This bug was fixed by making the thread sleep before performing an operation on a view.
- For the notifications, everything worked well, however when testing with other people, nothing seemed to function. The problem was that other people were on higher Android versions that required notification channels to be used. After implementation of that, things worked for everyone.