FACULTY OF SCIENCE AND ENGINEERING SEMESTER 2, 2018

IAB330: Mobile App Development

Assignment 3: App Prototype

Due Date: Friday, 2nd Nov 2018, 11:59 pm

Assignment submission as a team through Blackboard

Weight: 50%

You must sign below. By signing this form, you agree to the following:
We declare that all of the work submitted for this assignment is our own original work except for material that is explicitly referenced and for which we have permission, or which is freely available (and also referenced)

The assignment shall be conducted in a team of 3-4 students, each team member must sign, as it is a formal agreement that represents that everyone is contributing to the whole assignment.

Team Member Details		
John Tang	n9731913	[illegible]
David McClarty	n9939768	[illegible]
Fredrik Forsell	n10297057	[illegible]
Ivar Sandvik	n10297154	[illegible]

Repository Download Links

First repository: https://bitbucket.org/n9713913/iab330-scruff/commits/
Second repository: https://bitbucket.org/n9713913/ruff/src/master/
(**Download this one**) Final repository: https://github.com/JohnT64/Ruff

Dependencies and Other Important Information

There is a possibility that the application will ask for certificate credentials. Below are these credentials:

Alias: Ozwwegian

Password: 123456

Full name: Oz Norge

Organisation Unit: Ruff Limited Co.

Organisation: Ruff Limited Co.

City: Brisbane

State: Queensland

Country code: AU

The following are packages and other technology the application depends on:

Xam.Forms

Newtonsoft.Json - to deserialize/serialize (decode/encode) the response from the DB server.

Xam.Plugin.Media - to add cross platform camera and gallery access functionality.

Xam.Plugin.Notifier - to add cross platform local notifications functionality.

sqlite-net-pcl - for testing the local database, and to cache images. Will be used to cache popular searches, and maybe gifs and videos, to improve application quality in the future.

Completion of User Stories

This section lists the user stories that were to be acceptably implemented in Release 1, with accompanying status of whether each was properly implemented.

1. Feature that allows the owner to input data about their animal, such as type, breed, age, weight, general information, etc. Pertains to use case 2, 3. Priority: High. Estimated points: 4. **Implemented: Yes**

2. Feature that allows the profile to be updated with images. Pertains to use case 2, 3, 4. Priority: High. Estimated points: 2. **Implemented: Yes**

3. Feature that allows users to search via a text bar. Pertains to use case 7. Priority: High. Estimated points: 2. **Implemented: Yes.**

4. Feature that allows users to search via an advanced search tool where they can specify multiple attributes covering a wide range of aspects. Pertains to use case 7, 9. Priority: Medium. Estimated points: 4. **Implemented: Partially. Only searching by breed, gender, and name was implemented. Searching by animal type, price, and other qualities were not implemented but would've been implemented in the same way.**

5. Feature that allows users to login. Pertains to use case 11. Priority: High. Estimated points: 2. **Implemented: Yes**

6. Feature that allows users to register. Pertains to use case 12. Priority: High. Estimated points: 2. **Implemented: Yes**

7. Feature that allows users to logout. Pertains to use case 13. Priority: High. Estimated points: 1. **Implemented: Yes**

8. Feature that places links to all the at the bottom of the screen in iOS, or on a left-side hamburger menu that can be opened in the top-left corner in Android. Pertains to use case 14. Priority: Low. Estimated points: 2 **Implemented: Yes**

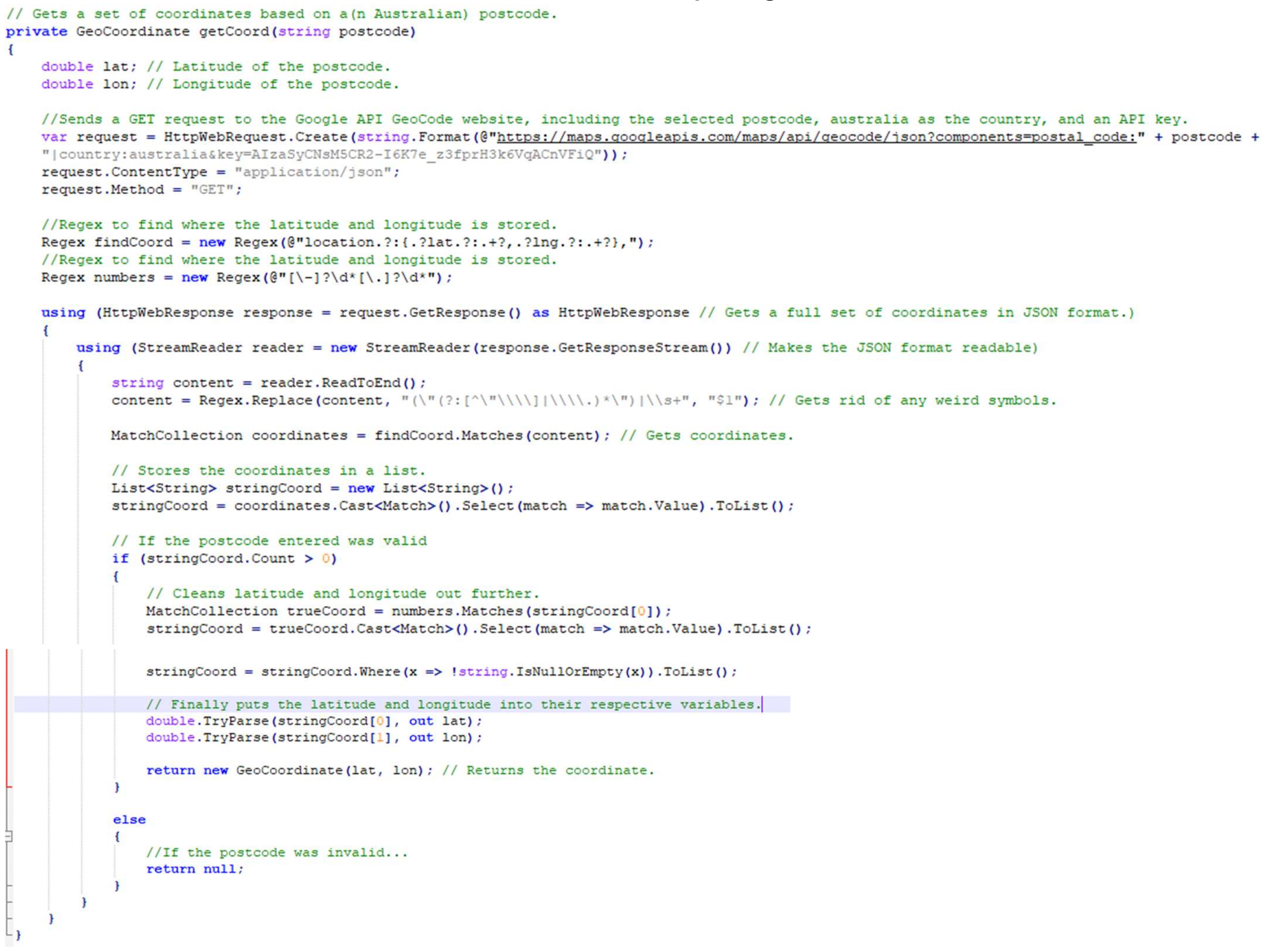
9. Feature that allows the user to change their password if necessary. Pertains to use case 16. Priority: High. Estimated points: 2 **Implemented: Yes.**

10. Feature that allows users to search via distance from the home of the owner to their home. Pertains to use case 7. Priority: Medium. Estimated points: 4 **Implemented: No.**

This feature would've been implemented by putting a list labelled "Distance" into the XAML file for the Search Page, the options of the list being "no limit", "less than 5km", "10km", "25km", and "50km". When the search button is pressed, the application would've checked what was selected in the "Distance" list. If the selection was something other than "no limit", the application would've considered the list when running a backend function called "AdSearch", a function that searches through the ad database.

This application would've used the following code shown in the images below to get the distance between the postcode of the user the ad belongs to, and the postcode of the user searching

(which would be retrieved either by geolocation or the user’s own stored postcode). The code would’ve made use of the “GeoCoordinatePortable” NuGet package.



```
// Gets a set of coordinates based on a(n Australian) postcode.
private GeoCoordinate getCoord(string postcode)
{
    double lat; // Latitude of the postcode.
    double lon; // Longitude of the postcode.

    //Sends a GET request to the Google API GeoCode website, including the selected postcode, australia as the country, and an API key.
    var request = HttpWebRequest.Create(string.Format(@"https://maps.googleapis.com/maps/api/geocode/json?components=postal_code:" + postcode +
"|country:australia&key=AIzaSyCNsM5CR2-I6K7e_z3fprH3k6VqACnVFiQ"));
    request.ContentType = "application/json";
    request.Method = "GET";

    //Regex to find where the latitude and longitude is stored.
    Regex findCoord = new Regex(@"location.?:{.?lat.?:.+?,.?lng.?:.+?},");
    //Regex to find where the latitude and longitude is stored.
    Regex numbers = new Regex(@"[\-]?\d*[\.]?\d*");

    using (HttpWebResponse response = request.GetResponse() as HttpWebResponse // Gets a full set of coordinates in JSON format.)
    {
        using (StreamReader reader = new StreamReader(response.GetResponseStream()) // Makes the JSON format readable)
        {
            string content = reader.ReadToEnd();
            content = Regex.Replace(content, "(\"(?:[^\"\\\\]|\\\\.)*\")|\\s+", "$1"); // Gets rid of any weird symbols.

            MatchCollection coordinates = findCoord.Matches(content); // Gets coordinates.

            // Stores the coordinates in a list.
            List<String> stringCoord = new List<String>();
            stringCoord = coordinates.Cast<Match>().Select(match => match.Value).ToList();

            // If the postcode entered was valid
            if (stringCoord.Count > 0)
            {
                // Cleans latitude and longitude out further.
                MatchCollection trueCoord = numbers.Matches(stringCoord[0]);
                stringCoord = trueCoord.Cast<Match>().Select(match => match.Value).ToList();

                stringCoord = stringCoord.Where(x => !string.IsNullOrEmpty(x)).ToList();

                // Finally puts the latitude and longitude into their respective variables.
                double.TryParse(stringCoord[0], out lat);
                double.TryParse(stringCoord[1], out lon);

                return new GeoCoordinate(lat, lon); // Returns the coordinate.
            }

            else
            {
                //If the postcode was invalid...
                return null;
            }
        }
    }
}
```

```
//code to put in the search function for location search
var animalCoord = getCoord(animalPostcode);
var userCoord = getCoord(userPostcode);

//final distance
var distance = animalCoord.GetDistanceTo(userCoord);
```

If the final distance was in the threshold specified by the “Distance” list, the ad would’ve appeared in the Results Page for the user to view. The distance would’ve also been displayed alongside the ad.

11. Feature that allows app notifications. Priority: Medium. Estimated points: 2.
Implemented: Yes

12. Feature that allows access to a user’s gallery. Priority: Medium. Estimated points: 4.
Implemented: Yes

13. Feature that allows use of the camera in-app. Priority: Low. Estimated points: 4. **Implemented: Yes**

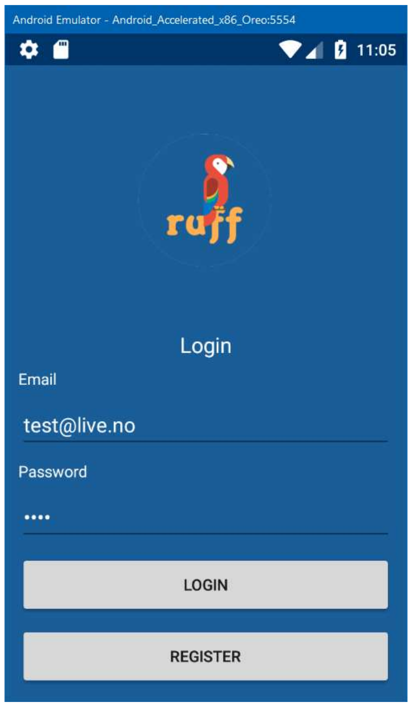
14. Feature that allows users to follow specific animals via a “feed” page which is regularly updated with any media posted on each followed animal’s profile. Pertains to use case 8. Priority: Low. Estimated points: 4. **Implemented: Yes**

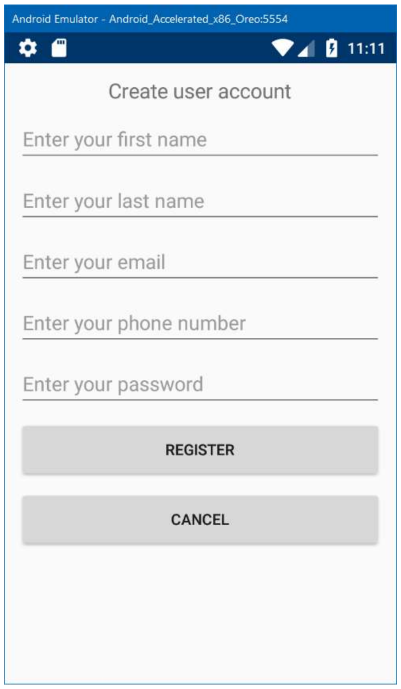
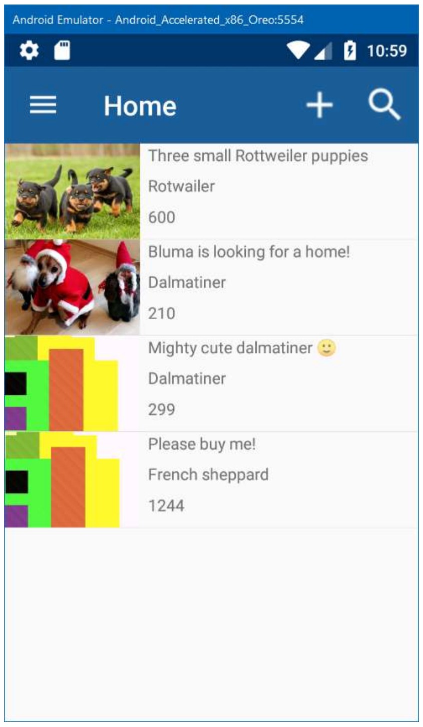
Additional Features

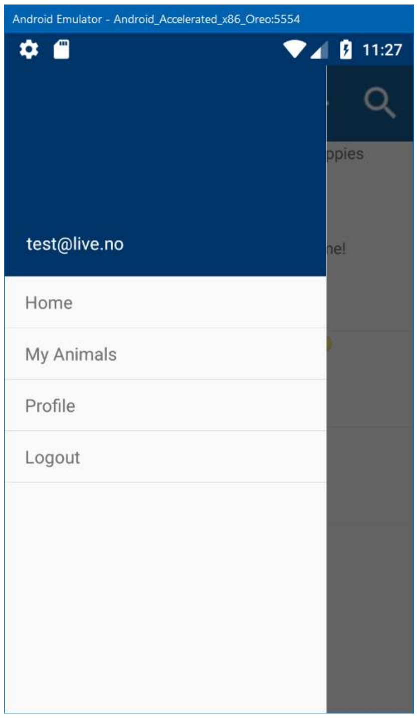
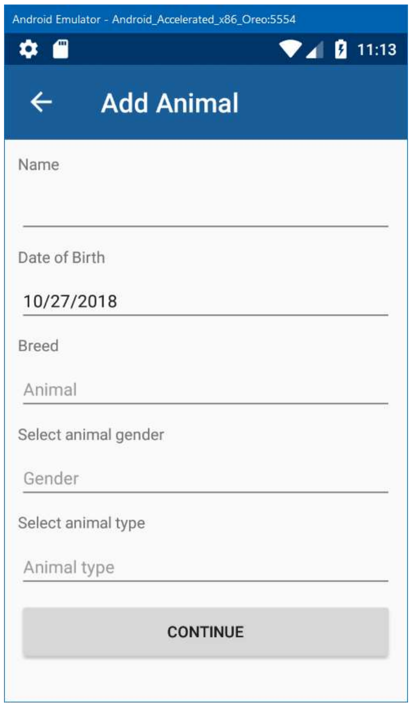
The application is already hosted on a public server in Norway, so anyone with functioning internet can theoretically access the program.

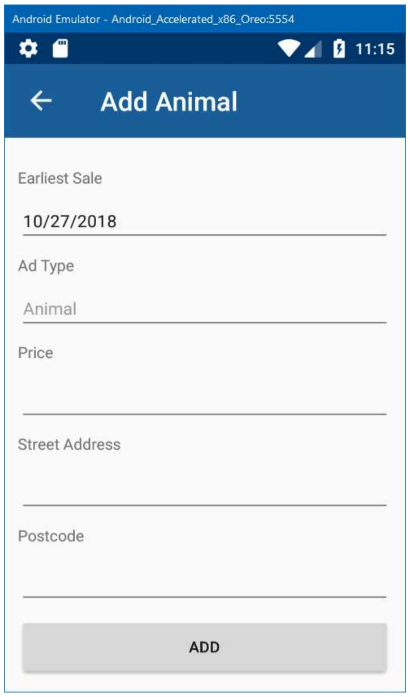
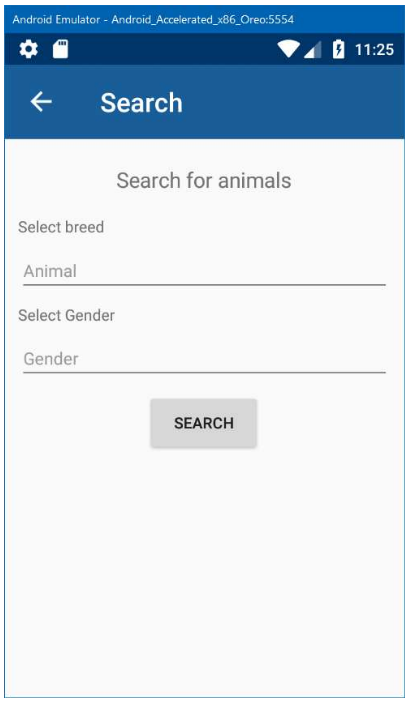
The ability for the user to change his profile picture is included in this release.

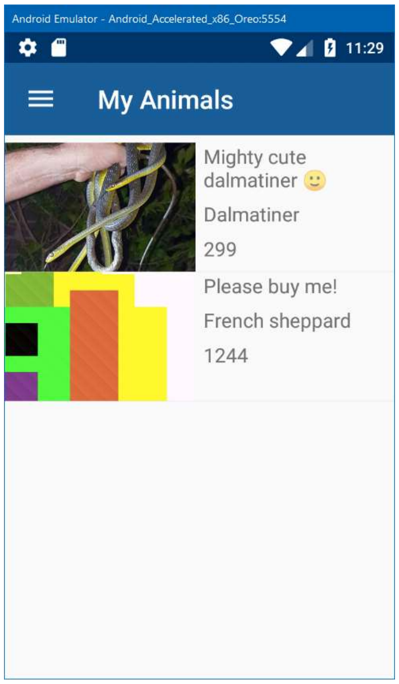
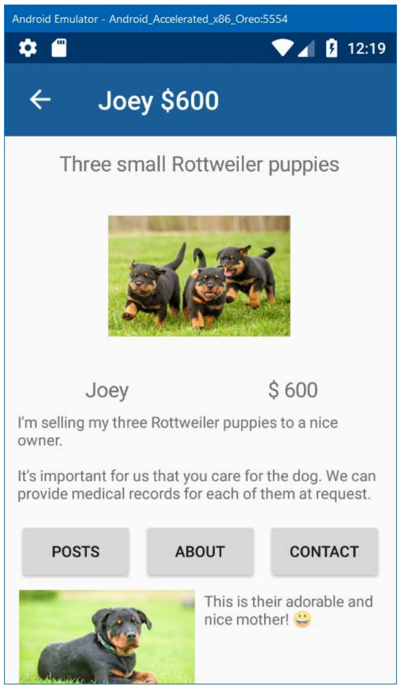
User Interface

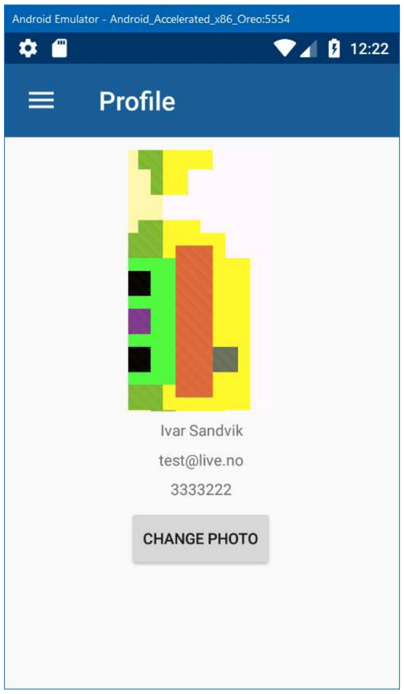
 Android Emulator - Android_Accelerated_x86_Oreo:5554 11:05 Login Email test@live.no Password •••• LOGIN REGISTER	The login page for the Ruff application. It contains fields for the username, and password. The login button will submit the credentials to the server to verify, and if it matches any known record, it will proceed to the home feed page. If any errors such as invalid credentials, a popup will show. The register button will take the user to a user account registration page.

 Android Emulator - Android_Accelerated_x86_Oreo:5554 11:11 Create user account Enter your first name Enter your last name Enter your email Enter your phone number Enter your password REGISTER CANCEL	The account registration page. This allows the user to use the app.Once the user has filled out the form with valid entries, the fields will be sent to the database server and stored. The two buttons confirm the form submission and cancel it.
 Android Emulator - Android_Accelerated_x86_Oreo:5554 10:59 Home Three small Rottweiler puppies Rotwailer 600 Bluma is looking for a home! Dalmatiner 210 Mighty cute dalmatiner 🙂 Dalmatiner 299 Please buy me! French sheppard 1244	This is the home feed of the application, which is where the user is taken to after logging in. Currently it shows all the ad results in chronological order of creation date. It is comprised of a listview containing the animal's profile photo, the ad's title, the animal's breed, and the asking price. Clicking on any of these ads will take the user to its individual page. On the app bar, there are actions; the first being the navigation drawer icon (3 horizontal stacked lined). The next is the adding animal button (plus sign) which allows the user to add a new ad. The last is the search icon, allowing the user to search for ads.

 Android Emulator - Android_Accelerated_x86_Oreo:5554 11:27 test@live.no Home My Animals Profile Logout	The hamburger menu has quick shortcuts to the most used functions of the app. “Home” takes the user to the home feed, “My animals” to a page containing a list view for all the animals owned by the current user, “profile” where a user can view their profile and make changes, and finally logout, which allows the user to sign out of the application.
 Android Emulator - Android_Accelerated_x86_Oreo:5554 11:13 Add Animal Name Date of Birth 10/27/2018 Breed Animal Select animal gender Gender Select animal type Animal type 	These two screens are both from the “add ad” process. It involves filling out fields related to the animal and the ad. These details are then sent to the central server to store.

 Android Emulator - Android_Accelerated_x86_Oreo:5554 11:15 Add Animal Earliest Sale: 10/27/2018 Ad Type: Animal Price Street Address Postcode ADD	
 Android Emulator - Android_Accelerated_x86_Oreo:5554 11:25 Search Search for animals Select breed: Animal Select Gender: Gender SEARCH	This is the search page. Currently the user can search by animal type (dog, cat, bird) and breed. In the future, additional filters will be implemented such as age, gender, and price range. Clicking the search button will query the central database and return applicable results.

 Android Emulator - Android_Accelerated_x86_Oreo:5554 11:29 My Animals Mighty cute dalmatiner 🙂 Dalmatiner 299 Please buy me! French sheppard 1244	The my animals page is similar to the home feed, but the results only show animals which are owned and managed by the currently logged in user. This allows the user to quickly make changes to their animals, as well as making posts. Clicking on any of the animals in the listview will navigate the user to the individual animal page.
 Android Emulator - Android_Accelerated_x86_Oreo:5554 12:19 Joey $600 Three small Rottweiler puppies Joey $ 600 I'm selling my three Rottweiler puppies to a nice owner. It's important for us that you care for the dog. We can provide medical records for each of them at request. POSTS ABOUT CONTACT This is their adorable and nice mother! 😀	This is the individual ad page. It contains the posts about the animal, as well as the basic and medical information. Finally there is the contact page which contains the owner information. On the header, the animal's profile photo is displayed, as well as the ad title, description, and price. The posts are a listview which contain the post text and the posted photo. The app is designed from left to right. Most users would be interested in the posts page. Once a user has become interested, they will navigate to the "about" page to find out more about the animal. Finally, if the user believes the animal is the right choice, they will contact the owner to arrange a deal.



Ivar Sandvik

test@live.no

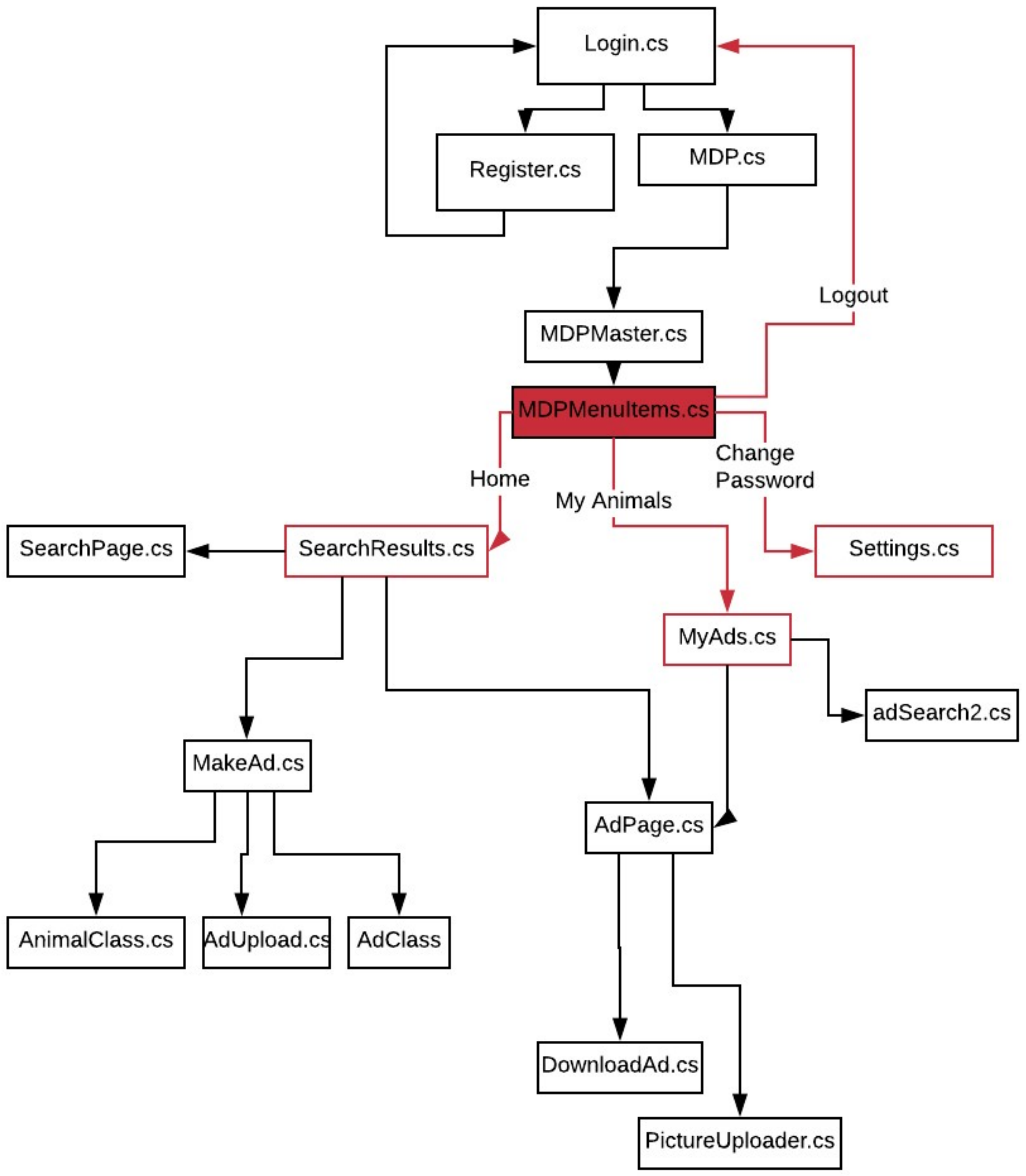
3333222

The profile page simply shows the profile photo of the owner, as well as their contact information. When navigated to, it downloads the profile information. A single button allows the user to change the photo.

Clicking on the “change photo” button will allow the user to select a photo or take one from their camera.

Further improvements include;
- Changing email and phone number button
- Show the seller rating (if applicable)
- Show the general location of the owner
- Seller reviews
- Show a list of their currently listed animals
- If they are part of a shelter/store, show those details
- Caching the profile information for faster load times

Class Call Graph:



The picture above shows all of the classes we have in our application, and how they are related to each other:

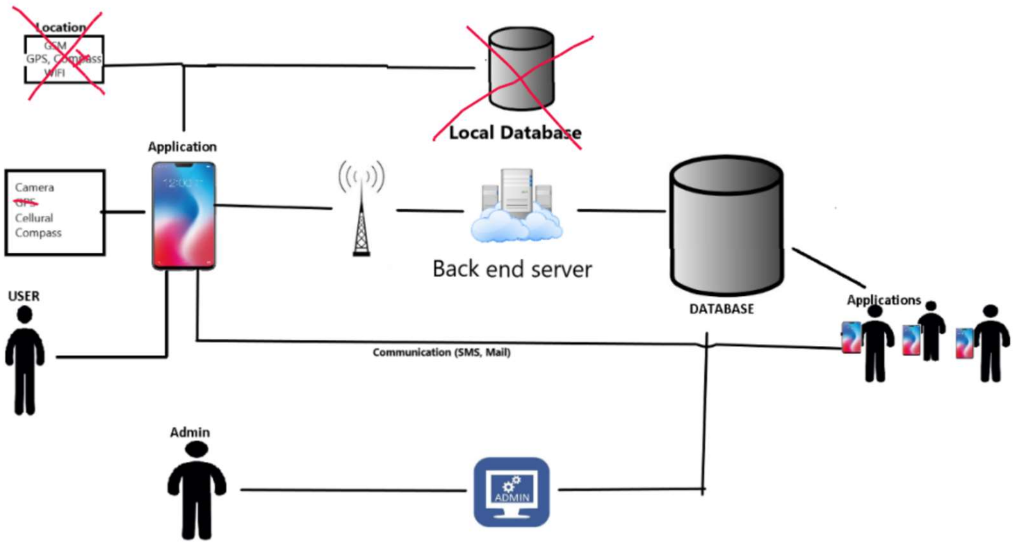
- Login.cs is the first page that shows up after launching the application. At this page you have to either login, or register a new account.
- After logging in the MDP is initialized, and the user can now explore the application freely using the side menu (MDPMenuItems.cs).

- Each of the buttons available in the side menu has a red arrow showing which class will be sent to. The logout button sends the user to Login.cs to create a new MDP session.
- SearchResults.xaml.cs has two buttons at the top bar. The search icon redirects the user to SearchPage, and the + icon redirects the user to the MakeAd page.
- If the user click on any ads from either the SearchResult or MyAds page, they will be redirected into the AdPage (of an individual animal).

Comparison between proposed and final software architecture:

When we were developing our application we wanted to make sure that the final product was connected to a functional database. Everyone that have the application installed on their device should also be able to view and add content to the database.

System architecture diagram:



Before we started coding and implementing features, we carefully planned how we wanted our final product to function. We did this by creating a system architecture diagram. The picture above shows the updated architecture diagram. The things we didn’t include in our final product is crossed out (Local database and Location).

As seen on the picture, there is not much difference from the original diagram. The **user application** is using Cellular and WiFi to communicate with a backend server. The server will then perform the changes (based on the instructions it is given) to the database.

We never implemented a method for finding the current location of the device. However, we created the code for it using the “Xam.Forms.Maps” plugin. The reason this wasn't implemented in our application was because we completely redid our database. The first version used a local database to store all of the data. Since the app didn't work with internet and multiple users, we ended up creating a new database on the cloud.

The 2 databases had different input data, which would result in us having to rewrite the logic. Since Location tracking wasn't an important feature for our application, we decided to focus on more important things instead. In the current build, location works by the user manually typing in their post codes.

Since our application is heavily reliant on data services. Fetching data from a backend server. At all times showed some limitations. We did not initially talk about caching being a necessity, but we know that our application will require some additional logic for not downloading data all the time. Since our server is located in Norway and the delay is high. We had to design the communication systems with robust feedback to the user. With every change the user will be able to know if this was saved etc successfully. So the backend is very robust in this sense and it handles that the connection is broken in middle of a transmission etc.

The database design is pretty much identical to the former design proposed in the first report for this application. There are some minor changes to the structure but these does not limit the functionality that we invented the system to have at project planning time.

As you can see on the diagram above, communication between users was supposed to be done through mail and telephone (outside the application). During the development we considered making our own messaging platform inside the application. We did not do this due to lack of time. It could be done in a few more development hours trough since our database was designed in order to support this kind of expansion.

Testing:
To ensure that our application was working properly, we used a couple of different methods for testing.

The method we used the most was manually testing. We installed the applications on all of the devices we had available. That includes 3 android devices and an android and Iphone simulator. We used these devices to check if the application had the same visual look across all of the devices. The elements had a small variation in size, but overall it looked pretty consistent. We also had to check if all features worked as intended.

Another method we used, was an inbuilt debug function in visual studio. We set breakpoints in the application while running it on an android device. This enabled us to check that variables were initiated and let us now exactly what happened when the application crashed. One example of where this method helped a lot, was when we couldn't get all of the advertisements to display. The content was correct in all of the variables, which meant that we had a problem was with the xaml and not the data fetching.

We also did a lot of testing with the backend services. The most efficient tool we found to do this was a JSON POST service. Were we could post a request (like the mobile app will do) and see the result from the backend server. This was more efficient compared to do it through the application.

Reflection on Learning

Several challenges were faced during development, notably **sharing the code** among the developers. As the development environment is dependent on many packages and files, pulling code from the repository required extra work to make the app run. This delayed the development of many features and even halted the progress of other developers due to these sudden incompatibilities. To resolve this issue, we minimised the working number of developers at a given time and scheduled “**commit times**” for each user to avoid merge errors. For future development, **Visual Studio Live** will be used to collaborate on the same files at the same time.

Another setback faced was the high latency experienced between the client and server, which resulted from the database location originating from Norway. The effect of this was the additional time required per test run. This was a necessity as by utilising this server, we were able to ensure the developers all had access to the same dataset and APIs available, in order to guarantee consistent testing.

Another testing related issue was the limitation of testing on Apple devices as the only capable machine on the team was a MacBook, which meant that the iOS version was neglected during some phases. This resulted in 90% of the testing being done on the Android version, with only the one developer testing the iOS version on an emulator.

In the middle of development, the team realised the lack of planning had snowballed its effects, and as a result, became behind schedule on feature implementation. For the next release of the application, more planning and scheduling will be done to ensure deadlines are set and achieved, and to set tasks more suitable to the timeframe. Another difficulty faced was the vague tasks assigned, which had no acceptance criteria to show completion. For future development, tasks will be more detailed in its description, and include acceptance criteria to be definitive in its requirements.