CHALMERS

360World
A mobile app for creating and sharing 360° images

Bachelor of Science Thesis in Computer Science and Engineering

OSCAR BOKING
JONATHAN HELSING
AMAR KULAGLIC
JOHN LIND
NEBOJSA MIHAJLOVIC
ANNA PETERSSON

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2017

Bachelor of Science Thesis

360World
A mobile app for creating and sharing 360° images

OSCAR BOKING
JONATHAN HELSING
AMAR KULAGLIC
JOHN LIND
NEBOJSA MIHAJLOVIC
ANNA PETERSSON

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
University of Gothenburg

Göteborg, Sweden 2017

360World

A mobile app for creating and sharing 360° images

OSCAR BOKING

JONATHAN HELSING

AMAR KULAGLIC

JOHN LIND

NEBOJSA MIHAJLOVIC

ANNA PETERSSON

Examiner: Arne Linde

Department of Computer Science and Engineering

Chalmers University of Technology

University of Gothenburg

SE-412 96 Göteborg

Sweden

Telephone: +46 (0)31-772 1000

Images and figures are captured or created by the authors, if nothing else is stated.

Department of Computer Science and Engineering

Göteborg 2017

360World
A mobile app for creating and sharing 360° images

OSCAR BOKING
JONATHAN HELSING
AMAR KULAGLIC
JOHN LIND
NEBOJSA MIHAJLOVIC
ANNA PETERSSON
Department of Computer Science and Engineering,
Chalmers University of Technology
University of Gothenburg

Bachelor of Science Thesis

Abstract

This report details the construction of 360World, an Android based prototype for a social media platform, the purpose of which is facilitating the capture and location focused sharing of 360° panorama pictures. The project encompasses the development of an Android application, a database for storing user information and the interface between these, as well as the configuration of hosting servers. The application makes use of algorithms from the OpenCV code library which are used to stitch multiple captured images into a single 360° panorama image; an image which can then be shared with the help of features that are typical for social media platforms, namely users and friends. 360World also has an innovative map-based way of visualising user profiles.

Social media as an industry has been on the rise since its inception, and picture sharing is one of its major features; 360° pictures, however, are still relatively fringe. In this project, we explore one possible way 360° pictures could be utilised in a social media setting. The project resulted in a working prototype with which users can create, and share 360° panoramic images via map-based image presentation.

Keywords: panorama, image stitching, image sharing, Android.

Sammanfattning

Denna rapport dokumenterar utveckling av 360World, en Android applikation vars syfte är att fungera som ett socialt medium där användaren kan skapa och dela 360° panoramabilder samt upptäcka bilder från andra användare genom en karta. Projektet innefattar utvecklingen av en Android applikation, en databas för lagrande av användarinformation, och gränssnittet där emellan, samt konfiguration av värdservrar. Applikationen använder algoritmer från kodbiblioteket OpenCV för sammanfogningen av ett flertal bilder till en 360° panorama bild; bilden kan sedan delas med vänner och andra användare. 360World använder sig även av ett innovativt kartbaserat visualiseringssätt för användarprofiler.

Sociala medier har varit på uppgång sen dess skapelse, och bilddelning är en av dess nyckelfunktioner; 360° bilder befinner sig dock alltjämt på utkanten. I det här projektet utforskar vi ett möjligt sätt 360° bilder skulle kunna användas på i en social media kontext. Projektet resulterade i en fungerande prototyp som låter användaren skapa och dela 360° panoramabilder via en karta.

Nyckelord: panorama, bildsammanfogning, bilddelning, Android.

Contents

1 Introduction

Internet and social media have been growing continuously, and are now larger than ever before. Today, almost half of Swedish Internet users visit social networking sites at least once a day, and send messages or post comments at least once a month [1]. This is a manyfold increase in comparison to 2012, when only one user in twelve visited social networking sites on a daily basis [2]. Mobile Internet usage has also increased sharply, and almost all Swedish Internet users now report using their mobile phones for online activities, compared to just over half in 2013 [1] [3]. Despite the large assortment of social media applications already available, new ones keep popping up, and there seems to be no sating the demand for such applications and their functionality; nor does there seem to be any reason to suspect that this demand will cease, as people will, probably, always be interested in new and improved ways of interacting with each other.

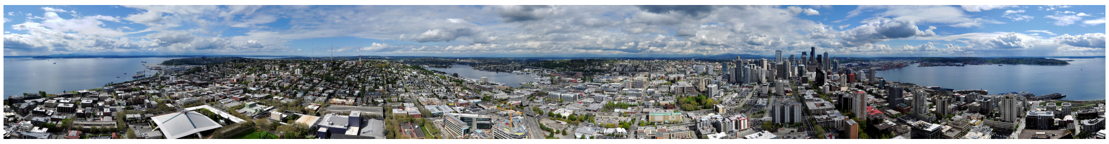


Figure 1: An example of a panorama with 360° field of view. From [4], CC BY-SA.

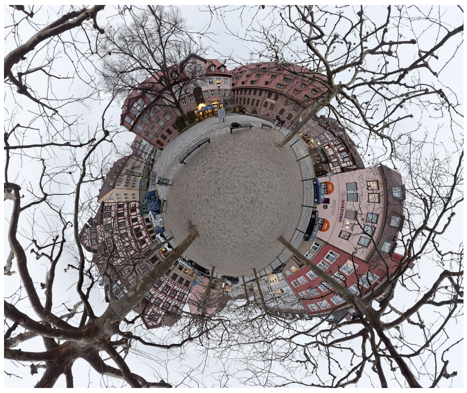


Figure 2: An example of a spherical 360° image, though these can only be viewed like a sphere in special viewers for these kinds of images. From [5], CC BY-SA.

The sharing of conventional images has been thoroughly explored by many social media platforms. The sharing of 360° images, however, has not been utilised to the same extent. A 360° image is typically defined as a panoramic photo with a 360° field of view (cylindrical), or a spherical image, viewed from the inside of the sphere, see examples in Figure 1 and 2. A 360° image allows for further immersion in the moment by the viewer, since it provides spatial information from all around the photographer.

When sharing an experience with others, the person sharing communicates what is going on at their location, and usually, where it takes place. Panoramas, as opposed to conventional pictures, enhance both these aspects of sharing by providing more context - both in terms of what is going on, and in terms of location. The location aspect of panoramic pictures makes them a natural match with geotagging, the purpose of which is to be able to inform users of where a picture was taken.

We intend to exploit the growing popularity of social media and the inherent synergy between panoramic images and locations by making a social network, where the pivot point is users taking 360° images and presenting them to other users through a map; both a general map where public images may be explored and map based user profiles. Utilising panoramic photos in a social media context is not without difficulty, as capturing panoramas is much more tedious than conventional photos, requiring much more effort from the user. This difficulty is likely the reason why panoramic photos are still relatively fringe, despite their potential. We believe however that the pros outweigh the cons, and that there is a way to make an application based on panoramic photos a meaningful addition to the social media landscape.

1.1 Purpose

The purpose of this report is to describe the development of 360World, a prototype for an Android based social media application. With 360World users should be able to create, view, and share geographically tagged 360° images in a way that emphasises the locational aspect, and is user-friendly.

1.2 Scope

This section presents a functional specification of the prototype, the limitations of the project scope, and the testing methodology employed. The scope and thus the feature specification for 360World have been defined by the members of the group as a result of that the project has no natural stakeholder. We have chosen to focus on honing our skills, both in terms of running a project, as well as overcoming the technical difficulties associated with developing a social media application and its peripheral infrastructure.

Table 1: The feature specification for 360World.

Function	Description
Create and view 360° images.	User is able to capture a panorama by rotating the phone, and is able to view the panoramas, taken by themselves or others, in a similar way.
Share a picture and upload a picture to any or all friends or upload a photo publicly.	User is able to share their pictures with selected friends or make them public (for everyone to see).
Save images locally without sharing.	Local storage in case the user does not want to, or is not able to, share the picture.
Create an account, and log in/out.	User creates an account with which his or her photos are associated with.
Add and remove friends from their friend list.	User is able to add and remove friends to or from the friend list associated with their account.
View a user's profile.	User can look at other users' pictures by looking at the map-based profile associated with their account.
See a map that lets the user explore other users' panoramas by tapping on a marker on the map.	Photos are represented on the map by markers at the location they were taken, clicking a marker shows a thumbnail with additional information, which can then be clicked again to view the photo.
Choose between seeing only his or her friends' panoramas or only public photos in the map view.	The user is able to toggle between showing only markers associated with photos taken by friends, or all markers, i.e. both ones shared by friends and public ones.

The scope of the application has been broken down into the features described in Table 1. Entries 1-3 of Table 1 describes the features associated with the creation and sharing of 360° images; entries 4-6 describe the ones related to social media; 7 and 8 describe the ones related to the location aspect of 360World, i.e. the map view and the map based user profiles. The map view (feature 7) can be conceptualised as the 360World equivalent to the ”timeline”, or ”feed”, often found in social media applications.

For storing the user information related to the social media functionality, a database needs to be designed and set up, as well as servers for the database, image storing, and web service. Additionally, a GUI (graphical user interface) has to be designed and implemented for the Android application, containing among others things, the map and search functionality of the map view. Moreover, the database, servers, and the application need to be integrated into a comprehensive system. Image stitching algorithms also need to be implemented in order to provide functionality for creating 360° pictures. These different tasks are visualised in Figure 3.

Due to time and budget constraints, the following are left for future development:

1. Addressing the security issues associated with retrieving and storing user data on a server.
2. Implementing spherical 360° images.
3. Adding support for other languages in the app.
4. Creation of administrative tools.
5. Allowing landscape mode.
6. Adapting the application for different screen sizes as well as for tablets.
7. Allocating the amount of server storage space normally needed for these kinds of applications.
8. Notifications in the application as well as push notifications.
9. User settings.

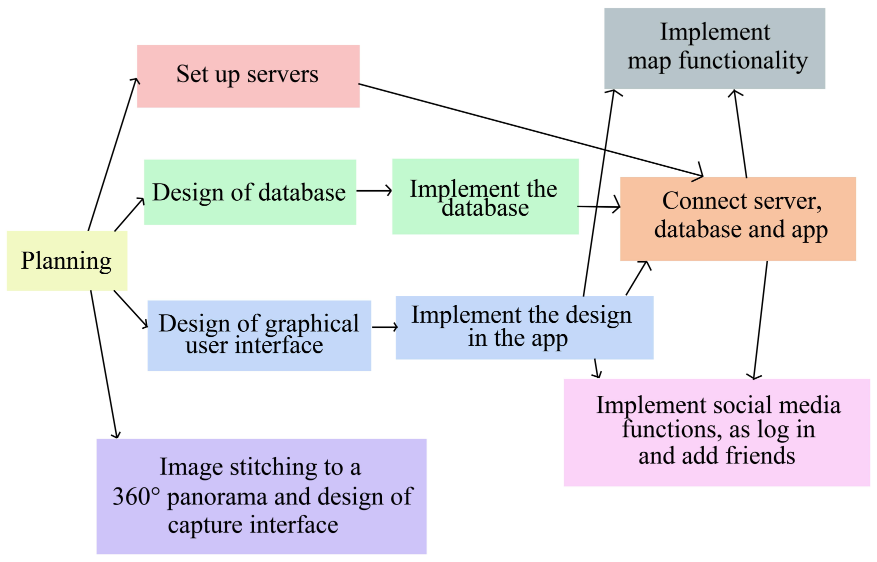

Set up servers
Implement map functionality
Design of database
Implement the database
Connect server, database and app
Planning
Design of graphical user interface
Implement the design in the app
Implement social media functions, as log in and add friends
Image stitching to a 360° panorama and design of capture interface

Figure 3: A flow chart showing the underlying tasks covering the scope of 360World.

2 Technical background

This section aims to give the reader insight and better understanding of the different areas that needs to be addressed in order to build the prototype, before reading the rest of the report. The Android operating system, user interface design, databases, server software, sensors in mobile phones and algorithms for image stitching are reviewed.

2.1 Android operating system

Android applications are written in the Java programming language and one of their main components are the activities [6]. It is important to have some knowledge of how an activity works in order to understand the basics of Android development. Activities are the main building blocks of Android applications and the key to receiving user input and displaying the various states that the program might have [7]. In other words, this is where the user interacts with the application [6]. For example, an email application might have three activities, or screens – one for showing new emails, one for writing emails and one for reading them [6]. An activity is implemented as a subclass of the Activity class and is started using an asynchronous message called an intent, which is created as an instance of the Intent class [6].

One vital aspect of activities in Android is their life cycle. An activity can be in one of four states: *running*, *paused*, *stopped* or *destroyed*. Calls to different event methods, for example `onCreate()` and `onDestroy()`, happen on state transitions. If the activity is in the foreground of the screen, its state is *running*. However, if for example the screen locks due to inactivity, the state changes to *paused* and the `onPause()` method is called. A visible activity, but with lost focus is also *paused*, which can occur for an example when the activity is behind another, transparent, activity. An activity is *stopped* when it is fully concealed by another activity, that is, it is hidden from the users but it still remembers all saved information. Finally, the activity can be *destroyed*, in which case the `onDestroy()` event fires. When making an Android application, it is crucial to understand how this life cycle works, since these methods likely will need to be overridden so that any developer-defined actions take place on state changes [8].

Another important part of application developing in Android is understanding fragments. A fragment can be seen as a ”sub activity” or a modular part of an activity since it is possible to have numerous fragments in one activity and reuse the same fragment in several activities, but it has its own life cycle. In contrast to an activity, a fragment is useless on its own because it has to be contained by an activity in order to be displayable [9]. The life cycle of a fragment and its methods are similar to that of an activity [9]. Fragments were mainly introduced to support flexible design on tablets and other devices with larger screens, making it easier for the developer to work on applications for both small and large screens [9].

An Android application also requires resource files for the visual presentation of elements, such as graphics or the layout of the user interface of an activity or fragment, which is defined using an XML file. Resource files can also define menus, themes and styles, strings, and colours. Having this kind of data in separate files makes it easier for the developers to alter the program to fit different device configurations [6].

2.2 User interface design

To create a prototype that is user-friendly, one has to understand some of the basic principles of interaction design. Interaction design is the study of how a user interacts with a product and not only about aesthetics in the design, but more importantly about understanding users and cognitive principles [10], [11].

During the design process, one can use fictional individuals, so called personas, to represent major target groups who are hypothetical users of the application. The usage of personas facilitates the design process, due to it being better to focus on the largest target group, rather than designing for everyone [12].

The traits of a product with high usability are efficiency, effectiveness and user satisfaction when completing tasks [13]. There are several methods used to improve the usability of a product. One of them is called *interface metaphors*, which is used to give the user an understanding of what new actions do by relating them to already known actions [14]. Another method is to use so called *design patterns* - structural and behavioural features which can be seen as descriptions of best practises - when it comes to designing for specific tasks [15]. This leads to interfaces that are easier to understand and use [15]. A specific design pattern cannot be used in every situation since they are created with a specific functionality in mind [15].

Design patterns are created with many different factors in mind, two of these being *flow* and *excise.* Flow is when users have full focus on their task while not being aware of their surroundings. While the user is in a state of flow, they can lose track of time, be extremely productive, and are less likely to get distracted by their surroundings. For the user to be able to reach a flow state, the interface of the application has to promote flow. This is done by reducing something called excise, which is unnecessary work that the user has to complete in order to perform their task [11].

There are many methods that can be used to detect usability problems in user interfaces. One of them is *heuristic evaluation*, which is when the interface is evaluated by a group of experts who follow some principles regarding usability. These principles are called heuristics and give answers to different aspects of the design [16].

2.3 Databases

A database is a dynamic collection of information which typically exists over a longer period of time. The term database more commonly refers to a collection of data, which is managed by a database management system (DBMS). The purpose of such a system is to [17]

- Allow for the creation of new databases with user specified data structures using a data-definition language.
- Allow users to query and modify the data using a so called query language.
- Enable the storage of large amounts of data and provide efficient access.
- Provide durability, e.g. sufficient recovery after a failure.
- Control interactions between users and ensure that no operation only partially finishes.

2.3.1 Relational databases

The most widely used database systems today are *relational database management systems* (RDBMS) [17]. A relational database model uses two-dimensional tables, or "relations", which are sets of unordered tuples, whose fields are filled with information according to the type specified by the relation schema. As an example, a relation schema might be

$$Users(\underline{username : TEXT}, password : TEXT, email : TEXT)$$

which describes the structure of a table called *Users* that has three columns, *username*, *password*, and *email*, all of them of containing information of a string data type. In the example above, the column *username* is a so called *primary key*, here denoted by underlining the column name, which means that it *uniquely* identifies all of the other columns in that row. In other words, if one know the value in the username field in a row, one also know the values in the password and email fields for that row. When the primary key column p of table A is a column in another table B, it makes for what is called a *foreign key* f for that table; any row in B must have a value in f, which exists in the p column of A. If a primary key spans across multiple columns of a relation, it is referred to as a *compound primary key.*

Structured Query Language (SQL) is a standard language used to create, query and manipulate databases. It consists of statements which fall into one of the following classes [18]

- Data Defintion Langage (DDL) - Used for defining the data structure of the relations.
- Data Manipulations Language (DML) - These statements perform queries, inserts, deletions, updates, etc.
- Data Control Language (DCL) - Used to restrict and grant rights to users.
- Transaction-Control Language (TCL) - Statements that allow changes made by DML to be grouped into logical transactions, which can be rolled back, etc.

2.3.2 Relational database design using an Entity-Relationship approach

A crucial first step in designing a database is identification of the information that needs to be stored and its interrelations. This is commonly done using the *Entity-Relationship Model*, meaning the compartmentalization of the data into *entities* and *relationships*, which results in an *Entity-Relationship diagram.* This diagram can then be translated into relation schemas (tables) using a set of mostly mechanical rules [17]. *Users* is a good example of an entity, while the fact that users can, for an instance, be friends with other users is a relationship. Entities are typically drawn as rectangles while relationships are represented as rhombi, and if two entities are in a relationship, they must both be connected by lines to that relationship. Binary relationships are the most common and will be the only type discussed here [17]. An entity has a set of attributes (relationships can have them too), which can for example be *username* or *password.* These are drawn as ellipses connected to the entity/relationship they belong to. An attribute is a primary key (or part of a compound primary key) if its name is underlined. An important aspect of connecting entities to relationships is the degree of the relationship, *cardinality*, which gives the associations a more exact definition. Two entities A and B can be in a relationship R with one of the following three cardinalities:

- Many-to-many (M:M)
  All members of A can be associated with all members of B, which is denoted by connecting A and B to R with lines (or curves).

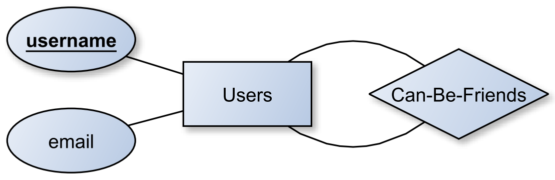


Figure 4: An example of an entity, *Users*, in a many-to-many relationship with itself. What this means is that each user can be in a friendship with multiple other users.

- One-to-many (1:M)
  Members of A can be associated with multiple members of B, while B can be associated with *at most* one member of A. This is denoted by adding an arrow to the connection on the 'one-side' entity of the relationship. When the arrow is drawn rounded, the above *at most* qualifier is changed into *exactly*.

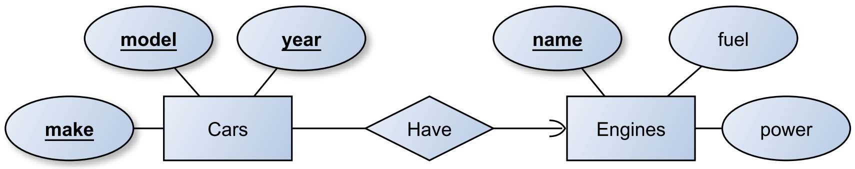


Figure 5: An example of an entity *Engines* in a one-to-many relationship with the entity Cars. Put into words, every car has exactly one type of engine, while that type of engine can be in many different cars.

- One-to-one (1:1)
  Members of A can be associated to *at most* one member of B and vice versa. Denoted by having arrows pointing to both A and B. One-to-one relationships are not very common because it is often better to combine the entities.

2.4 Server software

Server software is a class of software which is ran and managed on a computing server machine. These programs have the purpose of providing an interface to the computing power and data storage of the server machine to higher-level client services. There are many different types of server software used for a variety of applications, but the following are the types that are relevant to the prototype of 360World:

- **Web server software**

  The purpose of a web server software is to receive requests for and to deliver information stored on the server machine using HTTP or HTTPS. It allows the client to request execution of scripts, for an instance PHP, on the server, which are capable of interfacing with other services, such as database server software.

- **Database server software**

  Database server software provides clients with access to database services and functions. DBMS, such as MySQL or other database engines, commonly come with integrated server functionality.

- **FTP server software**

  Client services can get access to specified parts of the file system on a server machine through server software which implement the File Transfer Protocol (FTP). It is possible to have both upstream and downstream access, meaning that one can upload one's files onto the server machine, as well as download server content to one's device.

2.5 Sensors in mobile phones

In order to provide the user with guidance during the capture of a 360° image, one need to be able to detect when and how much the device is rotated, which is where sensors come in. Sensors are hardware components that analyse the surrounding environment. They can provide a number of different measurements such as temperature, pressure and distance. One example of a sensor, is the gyroscope, which measures angular velocity. Sensors exist in most mobile phones today, and have existed for quite some time. One of the first commercial smartphones with a build in gyroscope was the iPhone 4, which was released in 2010 [19].

In the Android sensor framework, there are hardware sensors, but also "software sensors". The software sensors are software abstractions, which combine measurements from one or more hardware sensor, sparing the programmer from having to manually combine these. An example of a software sensor is the so called "ROTATION_VECTOR", which combines measurements from the gyroscope, accelerometer, and magnetometer. This sensor provides more accurate data since it measures the three elements of the rotation vector of the device, instead of using the magnetic field and accelerometer. If possible, it uses all three common physical sensors when present (accelerometer, magnetometer and, if present, the gyroscope) [20]. This process is called sensor fusion and helps to provide more accurate data than using only one sensor. In some cases it might work better to use a high or a low-pass filter, however, it is usually the best alternative to use sensor fusion in terms of accuracy, completeness, and being more dependable [21]. In Android, sensors are divided into three categories: motion sensors, environmental sensors, and position sensors. The motion sensors measure acceleration forces on all three axes, the environmental sensor parameters like humidity and illumination, while the position sensors measure physical position (orientation) [20].

The sensors in the latest devices are highly accurate and consistent most of the time. However, they might be calibrated badly and since the measurements depend on external forces, data can be corrupted and misleading. Different methods can be utilised to mitigate errors and irregular spikes in the data, for an example the aforementioned sensor fusion. Other approaches can involve low or high-pass filters, or calculating the median of the neighbouring measurements. Additionally, accuracy and availability of sensors vary between devices, which is another issue that needs to be taken into account when working with sensors.

2.6 Algorithms for image stitching

To achieve a prototype which is able to create 360° images, the starting point is to study existing algorithms for image stitching, that is algorithms for combining multiple images into one single image. These algorithms are highly popular in computer vision and are for example used in areas such as image stabilisation in videos and the creation of satellite photos [22]. Because of their popularity, there are many different versions out there. For example, Brown and Lowe have developed an algorithm that is independent of the order in which it receives the images, can find multiple panoramas and also images that are not part of any panorama [23]. Moreover, Lin et al. have found an algorithm that is better at dealing with images with large parallax, something many other algorithms have problems with [24]. Liu et al., on the other hand, have developed an image alignment algorithm that can find aligning images among a set of images depicting different settings, which before has been difficult for aligning algorithms [25]. It is important to know how these algorithms work and their suitable areas of application in order to be able to select one that works well with the purpose in mind.

Many vital parameters are unknown in a fully automatic image stitching, for example the camera specifications, which images are overlapping and how much, differences in exposure, and changes in camera rotation or position between images. This means that an estimation of this data is required. Algorithms and methods for this are, among others, implemented in a library called OpenCV. Their pipeline for image stitching is very much alike the one described in *Automatic Panoramic Image Stitching using Invariant Features* by Brown and Lowe, mentioned above [26], [23]. This algorithm can be divided into two parts, image registration and compositing. The former aims to find how the images correspond to each other by finding so called features and how these correspond to each other. Compositing is the part when the images are combined into a panorama. An illustration of this pipeline can be seen in Figure 6. The different parts in this pipeline will now be described.

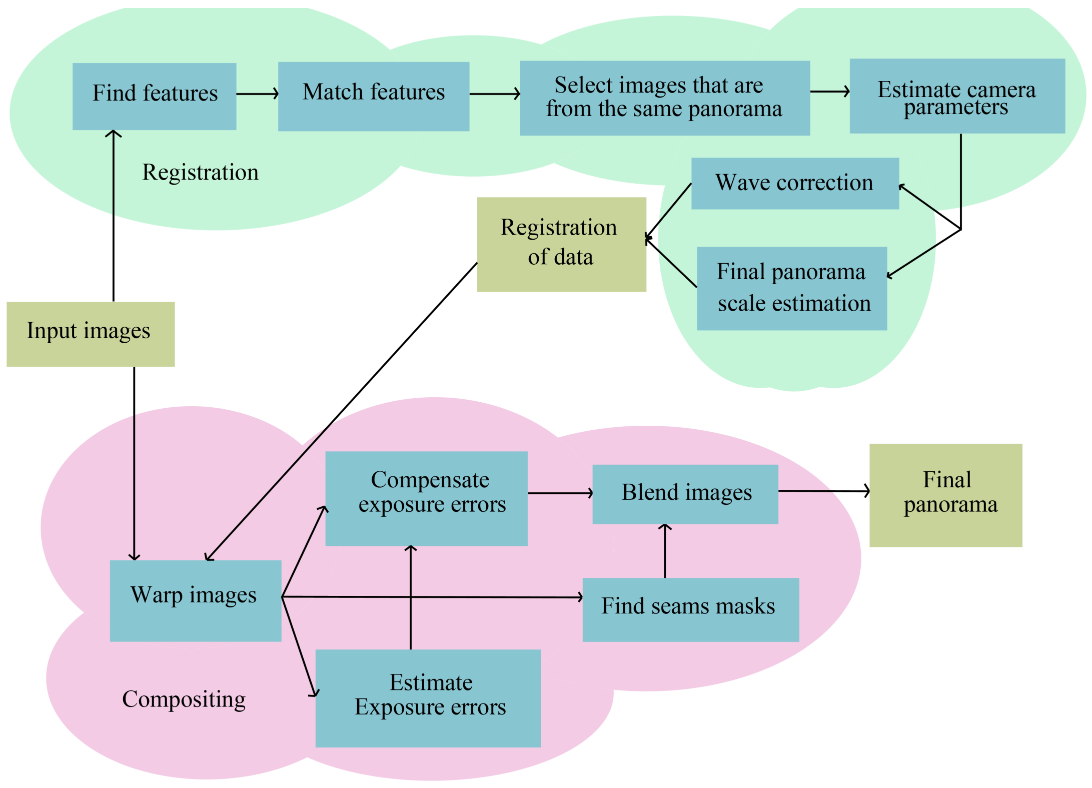


Figure 6: An flowchart showing the implemented stitching pipeline. The algorithm consists of two parts, registration of data and then compositing the images into the final panorama.

2.6.1 Feature detection

This first step is to find distinctive *features* in the images, that is, parts of the image that are unique. The features of different images can then be compared to determine if and how much the images are overlapping, since overlapping images will share some features. For this, three well-known algorithms are SIFT (scale invariant feature transform), SURF (speeded up robust features) and ORB (oriented FAST and rotated BRIEF) [27], [28], [29].

SIFT was presented by Brown and Lowe in 2004 and is an algorithm where the features found are unaffected by the images' scale and rotation. According to Brown and Lowe, the algorithm finds features that are highly distinctive since it is a high probability that one feature is accurately matched. Features are found here by computing the difference of Gaussians, an approximation of the Laplacian of Gaussian, and finding the local extremum over both scale and space. A local extremum is then a potential feature [27].

SURF is a scale and rotation invariant algorithm for finding features and is a faster version of SIFT. According to the developers Bay et al., this algorithm outperforms previous algorithms both in performance and computational time. Here, the Laplacian of Gaussian is instead approximated with Box Filter, as well as some other improvements [28].

ORB stands for *Oriented FAST and rotated BRIEF* since it builds on two algorithms called FAST and BRIEF [29]. ORB was developed by Rublee et al., and they show in *ORB: an efficient alternative to SIFT or SURF* that ORB indeed is faster than SIFT and SURF but still performs as well [29]. They had devices like mobile phones in mind when developing this algorithm and since ORB is not patented, as SIFT and SURF are, it can be used without paying [30].

2.6.2 Image matching and homography estimation

The next step is to match images that are overlapping in pairs and estimate how they are overlapping. If one assumes a pinhole camera model, the so-called *homography* describes how the two images are related. Assuming a camera model without translation we can compute the homography as

$$H_{ij} = K_i R_i R_j^T K_j^{-1} \tag{1}$$

where K_l is the intrinsic matrix of image l and R_l the rotational matrix for the same [31]. These homographies are what we want to estimate. Brown and Lowe do this by first finding the k nearest neighbours for each feature by using a k-d tree [23]. Then, for each image, they choose the m images that have the greatest amount of matching features with said image, because more is not needed to obtain a good solution [23]. Then, the homography between each pair is estimated using an algorithm called RANSAC [23]. RANSAC (random sample consensus) randomly samples a small number of matching features and then estimates the image transformation parameters [32]. This is repeated a number of times and the solution that has the greatest number of inliers is chosen [23]. The estimated homography can then be used to align the images [23].

2.6.3 Bundle adjustment and wave correction

After having matched the images together, we solve for all camera parameters using bundle adjustment. The means of estimating the camera parameters can also be called camera calibration and is the process of computing the intrinsic and extrinsic parameters of the camera that took the image [33]. The parameters that are specific to the camera, like for example focal length and lens distortion, are called intrinsic parameters [33]. The position and orientation of the camera are called extrinsic parameters [33]. The bundle adjustment is formulated as a non-linear least squares problem, which Brown and Lowe choose to solve by applying the Levenberg-Marquardt algorithm [23].

Then, the next step in the algorithm by Brown and Lowe is to apply an automatic panorama straightening [23]. By following the steps above, the final panorama will commonly have a "wavy" effect, this is as a result of that the real camera probably was tilted and not fully level when capturing the images [23]. This is corrected by applying a global rotation, making the "up-vector", that is the vector normal to the plane holding the centre of the camera as well as the horizon, vertical [23]. This is in OpenCV called *wave correction.*

This is the last step of the registration part of this image stitching algorithm, we now know which images that are matching and how as well as the orientation each camera had when taking the images. The next part is about combining the images into a panorama.

2.6.4 Projection

To be able to see the whole 360° panorama we need to project, or warp, the image, as one does with world maps. For panoramas which were made with a horizontal but not vertical rotation of the camera, a cylindrical mapping is fitting [31]. We can map the two-dimensional coordinates of the panorama into a three dimensional image with

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = R^{-1} K^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{2}$$

where K is the intrinsic parameters matrix and R the rotation matrix [31]. For a cylindrical projection, one can use the following (u, v) parametrisation

$$
\begin{aligned}
u &= \text{atan2}(X, Z) \\
v &= Y/\sqrt{X^2 + Z^2}
\end{aligned} \tag{3}
$$

where atan2 is the arctan function with two arguments and correct sign handling [31].

2.6.5 Seam estimation and exposure compensation

Seam estimation is a part of the algorithm that has been added and implemented by OpenCV, but was not included in the one proposed by Brown and Lowe. The boundaries between the captured images in the final panorama are often called seams and to estimate these is the first step in blending the images together, as a blending algorithm usually needs the seams as input [34]. The aim of the seam estimation is to compute the boundaries such that visual artifacts are minimised [34]. The function to be optimised is usually a pixel-based energy function, for example, variations in colour across the seam [34]. One of the most popular algorithms to achieve this goal is the Graph Cuts algorithm [34].

The next task is to compensate for varying exposure in the different images, this is a common problem when capturing images for image stitching using an automatic camera mode. Different exposures result in a panorama with differences in brightness level across it. The process of exposure compensation is also called gain compensation. Brown and Lowe solves this by setting up an error function as the summation over all overlapping pixels of the gain normalised intensity errors and then minimising this with respect to the gains [23]. Another approach, presented by Uyttendaele et al., is a block-based exposure adjustment where each image is broken down into blocks [35]. For each block, they want to find a function that best maps, in luminance, this block to the combination of images overlapping it [35].

2.6.6 Blending

The last step in this image stitching algorithm is called *blending* and its aim is to blend together the overlapping images, making the seams less visible. This is a relevant step since there still can be visible seams even after the exposure compensation, caused by effects that are not modelled in this algorithm, as vignetting and parallax [23]. Two known algorithms for this are feather blending and multi-band blending.

Using feather blending, the colour of a pixel is computed using a weighted average value. Pixels near the centre of the image have a higher weight and pixels close to the edges have a lower weight. The weight of each pixel is computed using a distance map. The average value is then calculated as

$$
C(\mathbf{x}) = \sum_k \frac{w_k(\mathbf{x})\tilde{I}_k(\mathbf{x})}{\sum_k w_k(\mathbf{x}} \tag{4}
$$

where $w_k(\mathbf{x})$ are the weights and $\tilde{I}_k(\mathbf{x})$ the warped images [22]. This algorithm is, according to Szeliski, acceptable at blending over exposure differences, however, not so good at dealing with blurring and ghosting [22].

Another approach is multi-band blending, or Laplacian pyramid blending, originally developed by Burt and Adelson [36]. The images that should be blended are in this algorithm decomposed into several band-pass filtered component images [36]. Then the images are assembled again, but they are joined using a weight that depends on the wavelengths in that specific band of frequencies [36]. To implement this, one can use Laplacian and Gaussian pyramids, thus the name Laplacian pyramid blending [36]. This algorithm should provide smoothness as well as avoid ghosting and blurring [31].

3 Method

This section aims to explain the methodology used, i.e how the application, database and servers were implemented.

3.1 Implementation of the prototype

The prototype was made using Android Studio, which is the official IDE (integrated development environment) for Android [37]. The alternative to this would be using Eclipse, with ADT (Android development tools), which was the standard some years ago. Android Studio, however, is the up to date, official environment, and thus the best choice today. As for programming language, the prototype was written in Java, which is the official language for Android development; C++ was used for the image stitching.

For version control, Github was used [38]. This helped everyone in the group to keep their local project in sync, as new code and assets were added by other team members. Moreover, it also makes it easy to roll back the project to an earlier state if someone uploads bad or faulty code.

The team worked according to Scrum, an agile development methodology [39]. We assigned eight weeks for implementation, which were split up into four sprints of two weeks each. The workflow of a Scrum project is illustrated in Figure 7. The figure shows the product backlog, which contains user stories with possible features of the application, and how they, at the start of each sprint, were prioritised and the user stories broken down into smaller tasks. After each sprint, a sprint review was held where the state of the product, and the work of the group, was evaluated. To help us work according to Scrum Trello was used, which is an online scrum task board, and this is where the backlog was kept [40]. Scrum is meant to enable greater flexibility, and might save time when implementing an application, compared to a linear method, e.g. the waterfall method, which can often lead to longer project times. It does mean, however, that the initial phase where the application is defined will take longer than it would with a linear methodology [41].

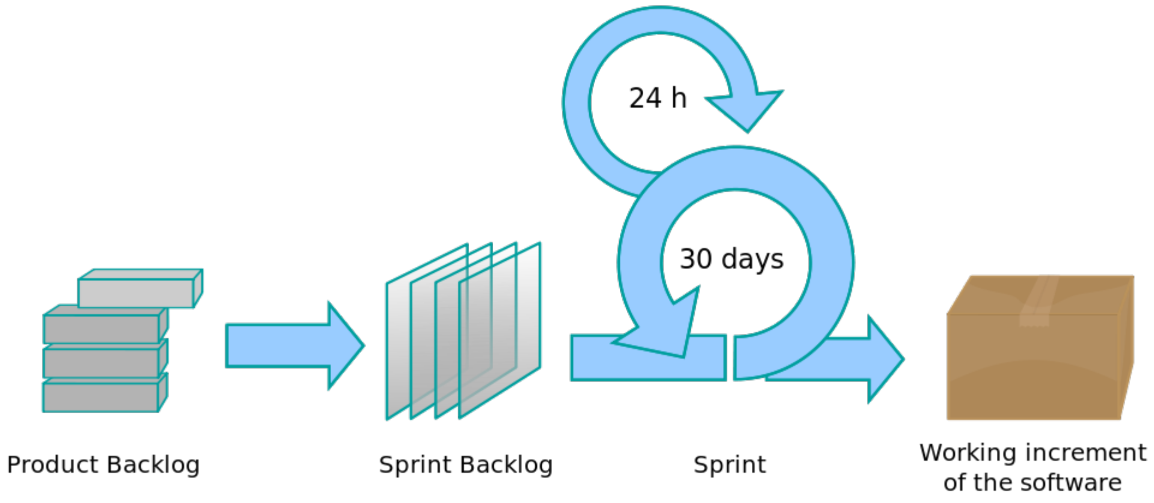


Figure 7: The Scrum workflow, showing how the product backlog is broken down into a sprint backlog for each sprint (30 days here, but our sprints were two weeks long) and how this creates a potentially shippable product after each sprint. From [42], CC BY-SA.

One key person in the Scrum workflow is the product owner. Traditionally, that person attends the sprint reviews and decides whether the team should continue with the development, or if the implementation phase can end. In our case, the implementation lasted for a fixed number of weeks, which led us to decide to allow for four sprints of two weeks each. There was also no specific product owner so we had to take on this role ourselves. The project owner decisions were made as a group, as opposed to, by one specific member.

3.2 User Interface Design

The commitment to user-friendliness compelled the group to spend significant time on interaction design. Despite there being some consensus within the field of interaction design as to what constitutes user-friendliness, it is ultimately subjective. And as such there is no definitive way of making an interface user-friendly, or of determining to what extent an interface fulfils this criteria. In this section the design process of 360World's user interface is described, and the design decisions taken explained.

3.2.1 The process of designing a user interface

The choice of method in the design process was based on the experience in the group, which meant that personas and sketches were used. The reason for using personas was that they provide a concrete representation of a potential user according to the target group of the application, as stated in Section 2.2. Therefore, the first step of the design process was deciding on a target group for the application. After group discussion and brainstorming it was concluded that the people most likely to appreciate an application such as 360World would be people between the ages of 20 and 40 years old, who are interested in travel, photography and social media. The design process proceeded with the creation of personas representing this group. These personas can be found in Appendix A. The hypothetical needs and desires of the personas, along with the suggestions and thoughts of the group, were then taken into account during the next step of the design process, in which a low-fidelity sketch was made; the purpose of the low-fidelity sketch was to concretise and visualise the ideas that had been generated with the help of the personas and group discussions.

The next step was to create a high-fidelity sketch and prototype based on the low-fidelity sketch. The prototype was created in order to showcase how the final prototype application would behave when interacted with. It is important that there are some ideas of how the general layout of an application should be before creating a high-fidelity sketch and prototype since these changes require more effort to change in a high-fidelity sketch and prototype than in a low-fidelity sketch. The insights gained from the low-fidelity sketch was used in the creation of the high-fidelity sketch and prototype. Our high-fidelity sketch and prototype included colours, icons, labels, etc., and adhered to the Android application guidelines – based on the design language *Material Design*, developed by Google – for interaction and visual design. One of Google's intentions when developing this design language was to make better use of the available space in the graphical user interface [43]. As mentioned in Section 2.2, interface metaphors help users understand design elements and are therefore used when applying this design language. Since this design language is recommended for all Android applications, users are likely to already be familiar with the visual design, making it easier for them to interact with the application. Using this design language can be described as using multiple design patterns developed by Google, the advantages of this is described in Section 2.2. This prototype can be found in Appendix B and was used as basis for the implementation of the application. Then, to test the usability of 360World, a heuristic evaluation was performed by two members of the group, as explained in Section 2.2.

3.2.2 Design decisions

Overall, all of these design decisions mentioned above increase the flow as much as possible in our application, while at the same time decreasing excise. The main focus during the design process was to provide a clean interface, limiting the number of visible objects on the screen as much as possible. This became especially important when the design of the map view began; a map on a smartphone is already hard enough to interact with, even without other objects obscuring it. Because of this, a navigation drawer was chosen as the main means of navigating the application instead of, for example, a tab bar. The order of items in the navigation drawer was based on what was thought to be used most frequently. A tool bar was placed on the top of each screen, providing a place for the required functionality of each view, without obscuring any important information.

Because the default map colours did not match the theme of the application, its colour scheme was customised to better complement the main colours of the GUI. Additionally, the amount of detail, such as small roads, were limited as they were deemed unnecessarily distracting. In accordance with previously described design philosophy, markers on the map were designed to be as clean as possible, avoiding information such as descriptions or titles. Information regarding the origin of a marker is relayed to the user with differing colours – green for photos posted by friends, red for public photos and blue for own photos.

A users' shared images are displayed in two different ways, either via the feed, or the map-mode of their profile. This provides the viewer of the profile with a locational overview of the users image collection, as well as a convenient way of viewing their entire collection in a list.

The friend list was divided into sections, alphabetically sorted after the first letter in the username. Additionally, an alphabetical scroll bar allowing more direct access to a given section. Both these features make it easier for the user to find whom they are searching for.

3.3 Database design

The initial design stage of the underlying database of 360World consisted of brainstorming sessions and sketching early versions of entity relationship diagrams (ERD). It was important to define what data would need to be stored and handled to best support the features specified in Section 1.2. A couple of revisions of the early sketches yielded the final ERD, which can be found in Appendix C.

This was followed by a relatively straightforward translation procedure, where only a few translation rules were needed:

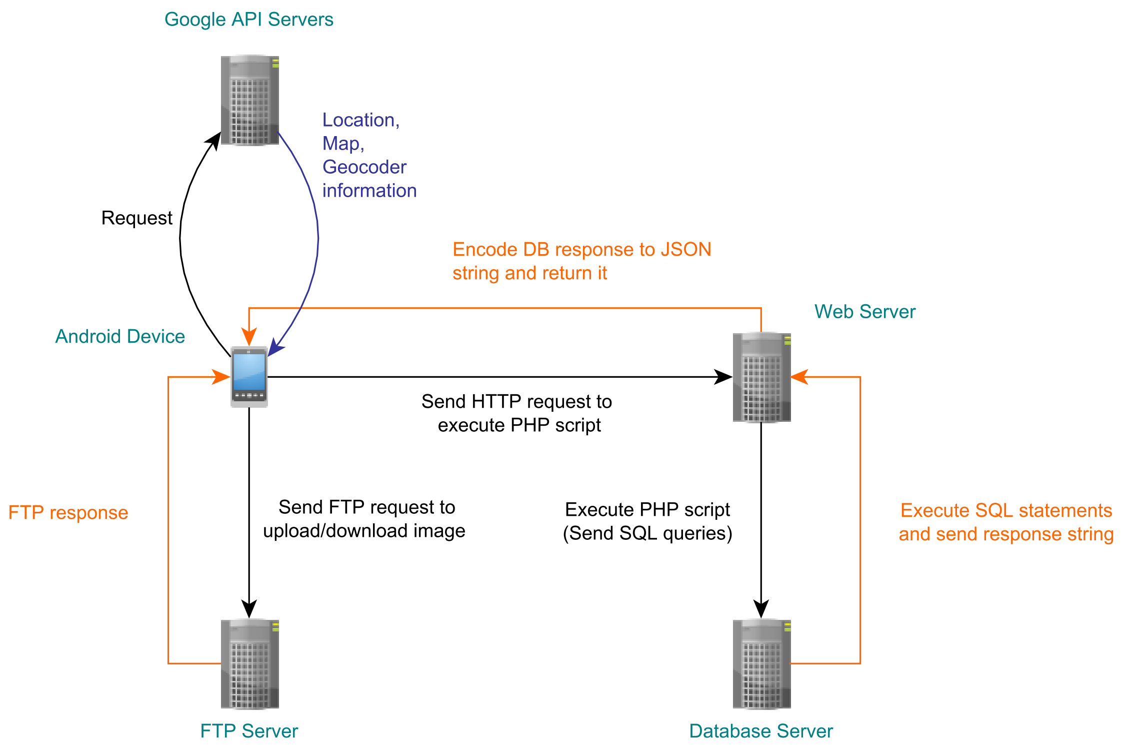
- Entities are translated straight into tables, their attributes making up the column names.
- M:M relationships are translated into tables with the columns consisting of the primary key columns of the connected entities (and any attributes the relationship might have, such as the date attribute of the *Uploader-Of* relationship).
- M:1 relationships are realised by making the primary key column of the entity on the 'one' side a foreign key column in the table of the 'many' side entity.

The translation resulted in a number of relation schemas, which could then be used as reference when constructing the database tables using the DDL statement portion of MySQL. The relation schemas are given in Figure 8, where *pkey* indicates a primary key and *fkey* a foreign key.

The four most popular RDBMS, which are Oracle, MySQL, Microsoft SQL Server, and PostgreSQL, all use some implementation of SQL as underlying software. Amongst these, MySQL and PostgreSQL are open source and free, but the first is more popular, which is why it was chosen for the 360World prototype application [44].

```
Users(username_pkey, password, email)
Friends(usera_pkey_fkey, userb_pkey_fkey)
        usera, userb -> Users.username
FriendRequests(usera_pkey_fkey, userb_pkey_fkey)
        usera, userb -> Users.username
ViewPermissions(user_pkey_fkey, imageid_pkey_fkey)
        user -> Users.username
        imageid -> Images.id
Images(id_pkey, uploader_fkey, date, description, longitude, latitude, public, views, likes)
        uploader -> Users.username
Views(imageid_pkey_fkey, user_pkey_fkey)
        imageid -> Images.id
        user -> Users.username
Likes(imageid_pkey_fkey, user_pkey_fkey)
        imageid -> Images.id
        user -> Users.username
Notifications(id_pkey, source_fkey, target_fkey, type, imageid_fkey(NULLABLE))
        source, target -> Users.username
        imageid -> Images.id
```

Figure 8: The relation schema resulting from the ER-diagram. Indented entries show what relation the foreign keys belong to.



Google API Servers
Location, Map, Geocoder information
Request
Encode DB response to JSON string and return it
Web Server
Android Device
Send HTTP request to execute PHP script
FTP response
Send FTP request to upload/download image
Execute PHP script (Send SQL queries)
Execute SQL statements and send response string
FTP Server
Database Server

Figure 9: The data flow between: FTP server, database server, Android device and Google API Servers. Note: While the figure shows three separate entities for the internal servers, they are actually hosted on a single machine owned by one of the project group members.

3.4 Communication between client and server

The three types of server software described in Section 2.4 were used on a personal computer in order to implement the device communication of 360World. More specifically, Apache HTTP Server is used as the web server, MySQL as database server and Pure-FTPd for the FTP server. In addition to the packets sent and received to and from the internal servers, the application also requires data, such as map information, location, and geocoding from Google API servers.

In order to query the database, the application must first send an HTTP request to run a PHP script on the web server that corresponds to the desired query. When the PHP code is executed, the query is sent to the database server through PHP's MySQL interface *mysqli*. The SQL statements specified in the query string are then executed and a reply string returned to the PHP script on the web server. This reply is then encoded into a JSON string and echoed, which is what the application receives to parse. The data flow of 360World is shown in Figure 9.

3.5 Map-based image presentation

This section describes the methodology used for implementing the map view, Google Maps API, and for retrieving the location of the device, Google FusedLocationProvider API, as well as converting between coordinates and locations, Geocoding API.

One key feature in 360World is the map mode, which lets the user explore the world map to find new and interesting images. For this a popular and widely used API, namely the Google Maps API, was used. With this one can easily define a map view in the layout which gives a world map where the user can navigate by scrolling, zooming and rotating the view. We also wanted the ability to easily place markers on and set the camera view to specific coordinates [45]. Apart from this, the design and theme of the map has to match with the theme of the rest of the app. Fortunately, this is also provided in the Google Maps API which can use a JSON (JavaScript Object Notation) file to associate different map objects with a colour, determine if some objects should not be drawn and change at which level of zoom different text labels should be displayed.

However, the Google Maps API does not include the function to convert from a location (that is, in the form of a string) to coordinates. Thus, the Geocoding API was also used, where this is provided. Apart from supporting location searching, it also supports reverse geocoding, which lets us retrieve a location based on coordinates. The search feature uses the geocoding in the map view, and the reverse geocoding is used to retrieve the location of an image in profile. Though, the Geocoding API has usage limits and only allows for 2500 requests per day and 50 requests per second. We have chosen not to address the problem of the large amount of traffic that usually comes from social networks since this is only a prototype.

In case the reverse geocoding does not find any location from the given coordinate, an algorithm which tries to find a location nearby instead is performed. The algorithm searches in a circle around the given coordinates, starting at a specified distance to the east of the coordinates and then going counterclockwise in a circle until one full revolution has been done. As mentioned above about the request limits from the geocoder, we therefore chose to only send a request every 45 degrees, which in the worst case searches for new coordinates a total of 9 times in the circle, including the first given coordinate.

It is important to have access to the location of a device in order for the image sharing in 360World to be done in any useful way. To this end, the Google FusedLocationProvider API was used, which is a library that wraps location update requests with both network and GPS as a provider. This means that if the GPS fails or is too slow, the API will automatically use the network architecture in order to attempt a coarse location fix.

3.6 Capture and display 360° images

Both for capturing the images used to create a panorama and for displaying the panorama, the sensors in the mobile phone have been used. These two will be explained in the following section.

3.6.1 Capture sequence

In order to be able to capture images which will be used to create the panorama, the camera of the mobile device needs to be accessed and used. The camera allows for a preview of the image and a photo can be captured by sending a signal to the camera hardware from the app. For regular pictures this would be all there was to it, since only one picture has to be taken. However, for multiple images that should be stitched into a panorama, the user needs to align the device in such a way that the photos are overlapping properly so that they can be stitched together. We thus needed to implement a capture sequence with guidance for the user, so the captured images can be turned into a panorama.

Since applications for creating panoramic images exist, we started by studying how they present the capture sequence. Some applications draw graphical dots as guidelines on the screen which move according to the rotation of the device. Other apps only represent the progress of the capturing sequence with a horizontal progress bar. Many of the applications also restrict the direction of the sequence to the right (clockwise). We decided to try and implement the moving dots since we considered them to be more user-friendly, as well as restrict the movement of the capturing sequence to the right.

While looking at existing apps, we also checked how many pictures they used in order to achieve a full 360° panoramic photo. Some apps need up to 35-40 images, while others only need around 20 images, in order to capture the full panorama. We experimented with different numbers of images and came to the conclusion that around 20 images were needed in order to achieve good results. Having 20 images means that for a 360° photo, the images will be captured 18° apart. The overlap gained from this depends on the camera, however, enough overlap would be gained for any camera device considering the amount of pictures.

In order to start the capturing sequence, the sensors of the device needs to be accessed. To accurately measure the rotation of the device we use the sensor type "rotation vector". This sensor combines data from available different sensors to give a more accurate result than if one were to only use raw data from for example the accelerometer and magnetometer [21]. The first use of this is so that we can check if the device is rotated to portrait mode (upright), which is needed to start the preview of the camera and allow the user to press the capture button. When the capture starts, the device registers the current horizontal orientation of the device as the origin angle, and then uses this during the sequence to get the difference in degrees relative to the first angle (that is, the progress).

During the capturing process, two surfaces are rendered on top of each other. In the back, the camera preview is drawn and on top of that, the graphics of the guideline dots are drawn. This is separated from the main camera, and the surface constantly has to communicate with the camera and sensors to receive updated measurements so that it can redraw the dot at a new location relative to the current orientation. In order to simulate dots around the space of the user, we display one dot at a time. A mathematical function is applied to get the offset the dot should have relative to the centre, given the current angle of the device and the number of already taken pictures. When the camera is oriented so that a dot is in the horizontal and vertical centre, the application starts a timer that after some time (1 second) checks if the dot is still in the middle. If that is the case, an image is captured and the surface view is made aware of the new angle it should target for the next picture. As mentioned in Section 2.5, the output is accurate, but can at times also sway. Because of this, we implemented an allowance for error of 2°. This will make the graphical dots not move excessively around the middle even though some sensor output might be bad. The sequence ends when the user presses the done button or when the total number of pictures (a full 360° image) have been reached.

3.6.2 Image viewer

There are some different approaches that can be chosen when making the image viewer. It can be a simple viewer, which only shows a single picture and does not allow for user interaction. However, we wanted to implement some features in the viewer that are related to panoramic images, as well as some navigational features. The user should be able to zoom the image, and since the image is likely to cover a large field of view, the user might want the image to loop (wrap) around so that it can be scrolled infinitely far to the left or right, by tapping and dragging.

Apart from this, a ”sensor mode” is implemented which lets the user look around the image by rotating the device. This is only allowed for images larger than the screen, since they would otherwise need to be stretched. While in sensor mode, the app assumes that the image is a 360° image. This means that the user will have to rotate the device 180° to get the image to scroll half its width, regardless of the images actual field of view. This is because there is no easy way of detecting if an image is looped seamlessly. To allow the sensor mode, we use the same sensor type as in the camera mode, the rotation vector. In the viewer, we use a canvas to draw three images side by side. When the image is scrolled to a side, all images are moved. If the user scrolls to the right, the leftmost image will get redrawn to the right of the current rightmost image, effectively creating the illusion of a stream of infinite images.

3.7 Creating a 360° image

In the following section, the implementation of the image stitching is described as well as how the memory management and image cropping was handled.

3.7.1 Image stitching

The goal with the image stitching was to make it reasonably fast, but still with a sufficiently good quality of the resulting panorama, thus in some cases, the fastest algorithms and not the best ones were chosen. Since the time span of this project was tight, it was chosen by the group to not implement an image stitching algorithm from scratch, mainly because it would have required significant resources and thus would have given us a higher risk of not having a working image stitching at the end of the project. Instead, one can try to find an existing library providing such an algorithm. Thus, the choice was not which algorithm that would suit best for this project and purpose, but which library that would. This limited the options considerably. Some libraries for computer vision that have support for image stitching are for example OpenCV [26], Halcon [46] and Matrox Imaging Library [47]. In the end, OpenCV was chosen because it is free, and widely used [48], providing us with a large community from which to get help should we need it. More specifically, the implemented algorithms found in the ”stitching” module of OpenCV were used. However, this ”stitching” module is not available in the version for Android Studio, instead, the version made for C++ was used. The C++ code was integrated into the Android Studio project using CMake, a tool made for this purpose. Two different versions of the image stitching were implemented, one easier and one more complex, to ensure that 360World would have this functionality, even if the more complex stitching would not work.

To begin with, the image stitching was implemented using the built-in stitching class in OpenCV called *Stitcher*. The pipeline for this algorithm is explained in Section 2.6 and illustrated in Figure 6 in the same section. In some parts of the pipeline different algorithms can be used, but the default parameters in OpenCV for this pipeline are, among others, to use SURF as feature detector, block-based exposure compensation and multi-band blending. The default parameters were used in all steps except for feature detection, were ORB was chosen instead, since this is supposedly faster and more adapted for mobile devices, as described in Section 2.6. As an example of the other parameters, was the default blender, multi band blender, not changed to the less advanced feather blender (as described in Section 2.6) since blending is an important part when making a panorama from images taken with a hand-held camera on automatic mode, where the risk is high for visible seams and artifacts.

After the first part was working, we tried to implement a more complicated version. Here, the individual algorithms which lead up to a panorama, as finding features, was used and the same pipeline as before. The default parameters in OpenCV was used, except for ORB as feature detector instead of SURF. The reason for implementing this version was mainly to gain more understanding of how the image stitching pipeline works as well as to test if it was possible to implement the image stitching "better", that is faster or with better quality in some way than in the built-in stitching class *Stitcher*.

3.7.2 Memory managagment

Issues were encountered when attempting to stitch many (up to 20) images together; the application did not have sufficient heap memory available to allocate and later process the bitmaps in the stitching algorithm. Multiple steps were taken in an attempt to address these problems, but the two most important were reducing the image size and storing the images in native memory instead of the application heap. The image size was half of the original captured image size. There was also the option of changing the colour format to 16-bit instead of 32-bit, but the quality impact would have been significant, so it was kept as a last resort option. The *Java Native Interface* (JNI) provides an interface between Java code and native C/C++ code, which allowed for storing of data into native memory and accessing it through the application Java code, an illustration of this can be seen in Figure 10. This was helpful because when data is stored in the native memory, it is not limited by the heap size given to the application by the JavaVM; theoretically, you have access to the entire RAM of the phone that is not currently in use. Not only that, but operations on native memory are also faster [49], which could be significant when discussing heavy algorithms such as image stitching.

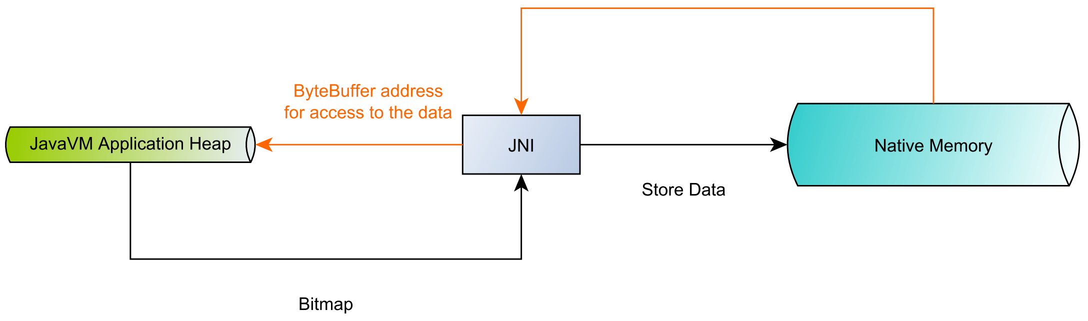


Figure 10: Illustration of the memory flow between JavaVM heap and native memory. The bitmap object is passed to the native environment, where it is stored as a sequence of bytes. An address to this buffer is then sent as a ByteBuffer response to the JavaVM environment.

3.7.3 Image cropping

Another problem to solve was the issue that the final panoramas had black borders after stitching, especially if the resulting image was not fully 360 degrees. The borders in the resulting images are highly irregular, as is shown in Figure 11 below.

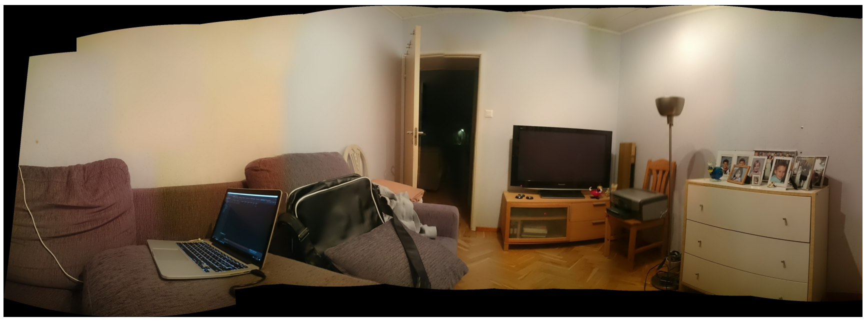


Figure 11: A final panorama before applying image cropping. The image shows a distinct irregular black border present after stitching.

A custom algorithm was devised for this purpose with the following steps at its heart.

- **Contour identification**
  This is achieved using an approach based on an algorithm proposed by Suzuki and Abe [50].
- **Quadrilateral approximation**
  After a contour curve is found, the number of points is reduced by running it through a version of an algorithm proposed by Douglas and Peucker [51]. Epsilon is tweaked until the contour is reduced to only 4 points, which are assumed to be the corners of an irregular quadrilateral.
- **Corner identification**
  The corners that the points represent are identified and a rectangle which fits inside of the quadrilateral is calculated. This rectangle is then used to crop the image.

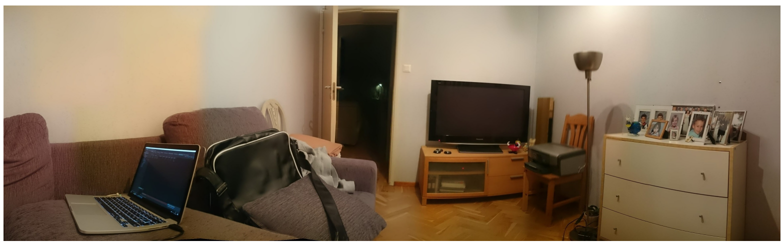


Figure 12: The same final panorama as in Figure 11, but after applying the described image cropping algorithm.

Figure 12 demonstrates an example of how the panoramas look after the cropping and shows the image in Figure 11 after it has been processed by the algorithm above.

4 Results

The project has resulted in a prototype, which is presented in this section. The application itself is presented in Section 4.1, followed by an evaluation of the image stitching in 360World in Section 4.2 and then the results of the heuristic evaluation is presented in Section 4.3.

4.1 The prototype

This project has resulted in a prototype application where the user can create and share 360° images and view both friends' and public panoramas via a map view. In this section, the views, navigation, and usage of the application are presented and explained.

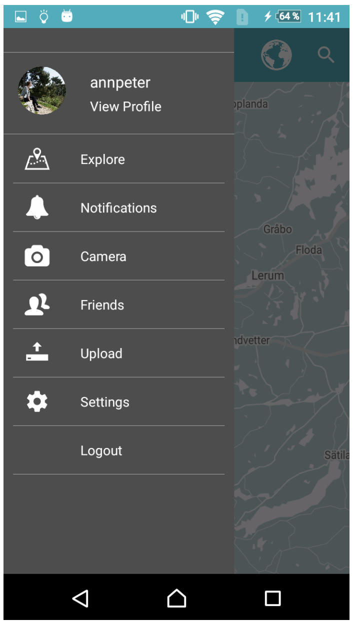


Figure 13: The navigation drawer of 360World with its main views.

To get an overview of the app, we start by showing the navigation drawer, see Figure 13. Here, the user can navigate to the main views *Explore*, *Notifications*, *Camera*, *Friends*, *Upload*, *Settings*, and to the user's own profile. The user can also log out from the app. The navigation drawer is available from all views, except for when capturing an image. It is accessed by swiping from the left edge of the screen, or by pressing the button in the upper left corner (not visible in Figure 13, but in, for example, Figure 15a). The main views show the following:

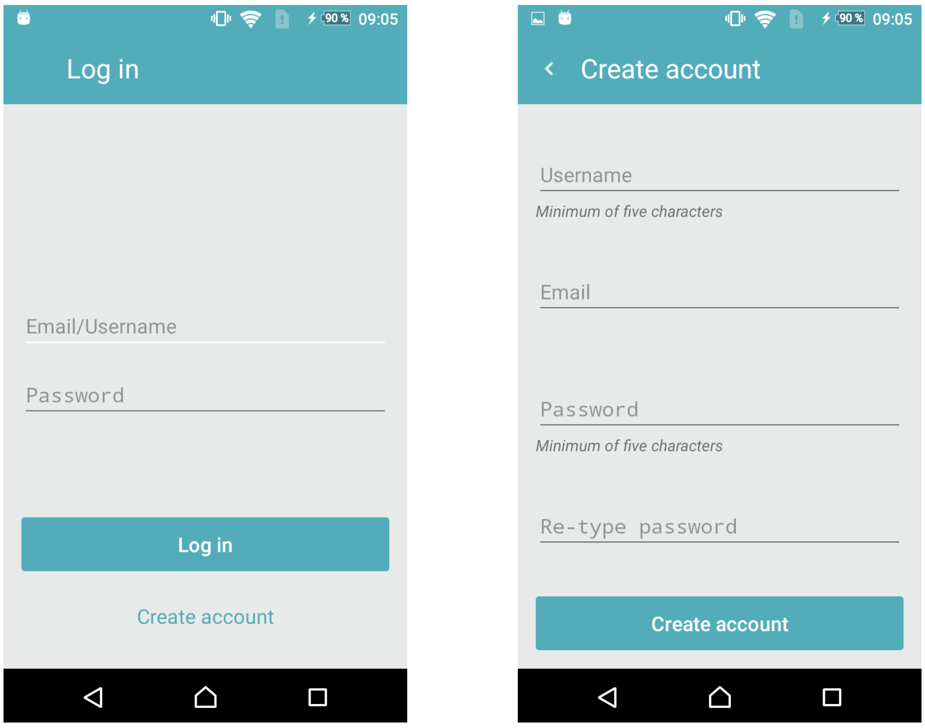
- Pressing **View Profile** will show the user's own profile.
- **Explore** contains a map with markers at the location where panoramas were captured, here the user can browse and explore panoramas captured by friends or other users.
- **Notifications** shows a list of the user's new notifications. The user would get a notification when a friend shares a panorama with him or her, as well as when the user receives a friend request. Though, as stated in Section 1.2, this is not covered in the scope of the project, and is left for future development.
- **Camera** is where the user can capture, and share or save 360° images.
- **Friends** shows the user's friend list. The user can also search for, and send friend requests to other users here.

- **Upload** is for sharing a panorama previously captured and saved on the phone.
- **Settings** would contain various settings, but as *Notifications*, it is not covered in the scope of the project.
- Pressing **Logout** will log the user out of the application, and return them to the log in view.

More thorough explanations of these views follow on the next pages.

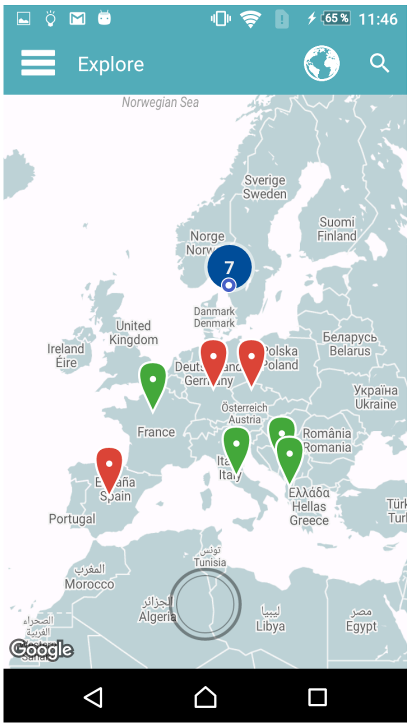
4.1.1 Log in and create account

If no user is logged in, the first view shown when starting the app is the log in view, showed in Figure 14a. Here, the user can log in by providing their username and password. They can also create a new account, by clicking on *Create account*, which takes them to the view in Figure 14b, where they then fill in their information. When the user tries to create an account, multiple checks are performed on the given information, one of which is making sure the username has not already been taken by someone else. If the information passes all the checks, a new account is created. If on the other hand, any of these checks fail, the user is notified of the error and how to correct it. Similar feedback is also given in the *Log in* view, if the user tries to log in with an invalid username or password.

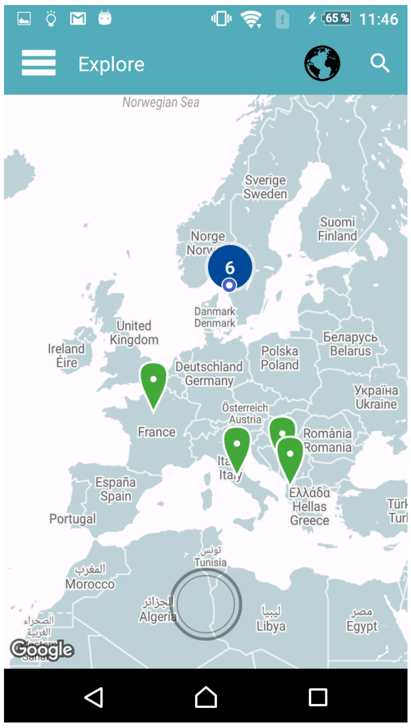


(a) Log in. (b) Create account.

Figure 14: Figures showing the views for log in and for creating a new account.



(a) Friends' and public photos.



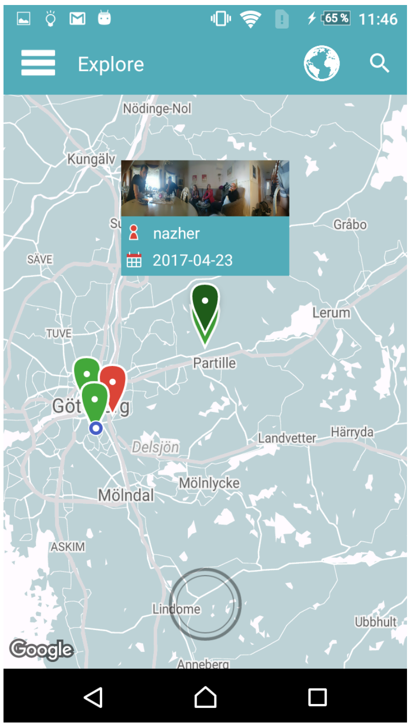
(b) Only friends' photos.

Figure 15: Figures showing the *Explore* view with markers showing friends' (green) and public (red) photos; the big blue markers are marker clusters.

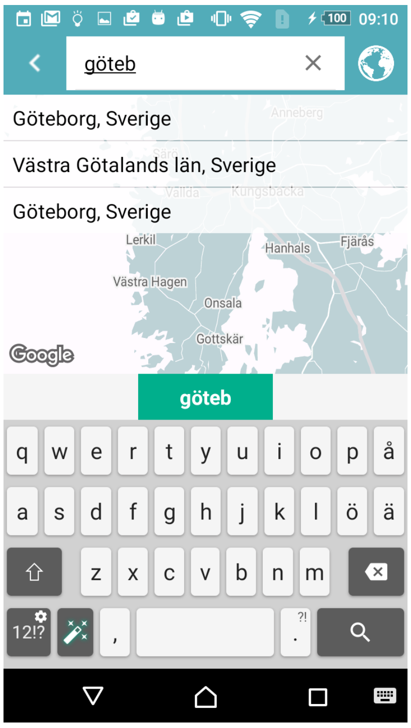
4.1.2 The *Explore* view - geotagged images on a map

If the user is already logged in, the map view (*Explore* view), shown in Figure 15a, is the first view displayed. This is also the view the user is taken to after logging in or creating an account. The *Explore* view consists of a world map with markers in different colours which represent uploaded panoramas, the markers are positioned at the location where the panorama was captured. Red markers represent public panoramas, green markers represent panoramas shared by friends of the user, and blue markers represent the user's own panoramas. The blue round marker (positioned in Gothenburg in Figure 15) represents the user's current location. The user can choose to either show all pictures they have permission to view (see 15a), or only those uploaded by the user themselves and their friends, by clicking on the globe button in the upper right corner (see 15b). The round button at the bottom of the view takes the user to the camera view, which can be seen in Figures 20 and 21.

When the user presses a marker, a window with a thumbnail of the associated image, name of the uploader, and the upload date is shown (see Figure 16a). Tapping this window starts the image viewer, displaying the associated image; the image viewer is shown in Figure 17a. Tapping on a map location not occupied by a marker makes any open thumbnail windows disappear. Tapping the magnifying glass icon in the upper right corner reveals the user to the search bar, with which the user can search for locations by entering text strings, shown in Figure 16b. Tapping one of the search results in the drop down list below the search bar centers the map around that location.



(a) After tapping on a marker.



göteb

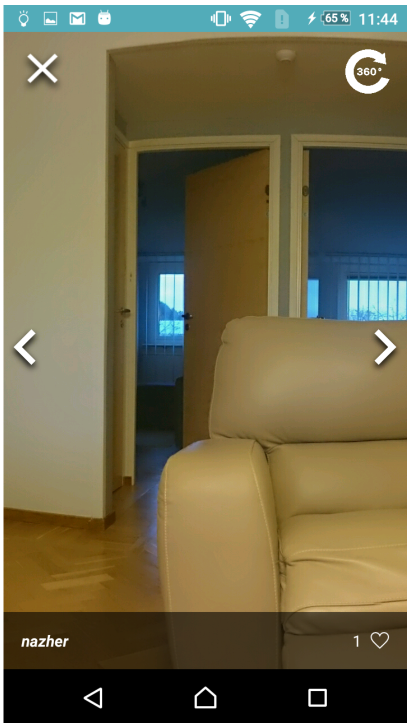
Göteborg, Sverige

Västra Götalands län, Sverige

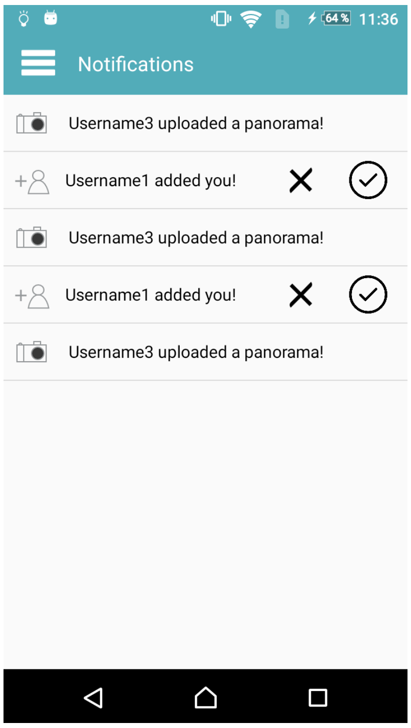
Göteborg, Sverige

(b) Location search.

Figure 16: Figures showing 'Explore' view after a user has tapped on a marker (left) and the location search.



(a) Image viewer.



Notifications

Username3 uploaded a panorama!

Username1 added you!

Username3 uploaded a panorama!

Username1 added you!

Username3 uploaded a panorama!

(b) Notifications.

Figure 17: Figures showing the app's image viewer for panoramas and the notifications view.

4.1.3 Image viewer for 360° images

The image viewer is shown in Figure 17a. The user can view the image in two different modes and can choose between these by pressing the button in the upper right corner. When in the mode shown in Figure 17a, the user shows the next part of the panorama by pressing the right or left arrow or by swiping on the screen. When in the other navigation option, the "sensor mode", the user rotates the device to show the next part of the panorama. When the viewer has reached the end of the image, it starts from the beginning - thus it can show 360° panoramas as a complete cylinder. At the bottom of the viewer, it shows the name of the user who uploaded it and how many likes the image has. To "like" the image, the user can press the heart.

4.1.4 Notifications

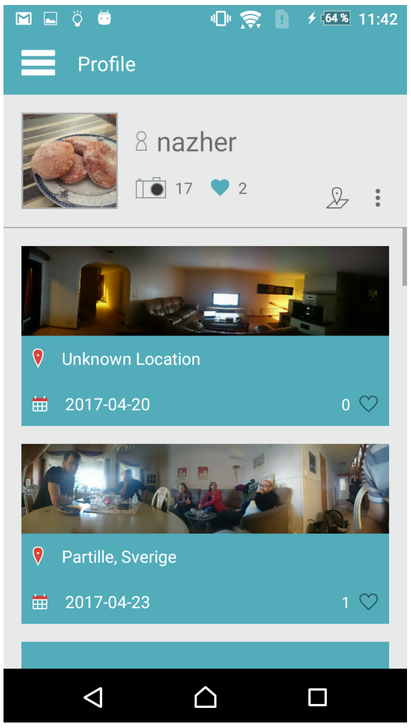
Figure 17b shows the *Notifications* view. Notifications and push notifications are, as mentioned in Section 1.2, not implemented in this project but the layout has been added to lay the foundation for future development. The idea is that a user should get a notification here as well as a push notification when a friend shares a panorama or when a user sends a friend request. Tapping the push notification should start the *Notifications* view. When pressing a notification about an uploaded 360° image, a view showing a map with a marker where the image was taken should start. Then the user can choose to press the marker and show the panorama. Pressing a notification about a friend request should show a view with that user's profile.

4.1.5 Profile

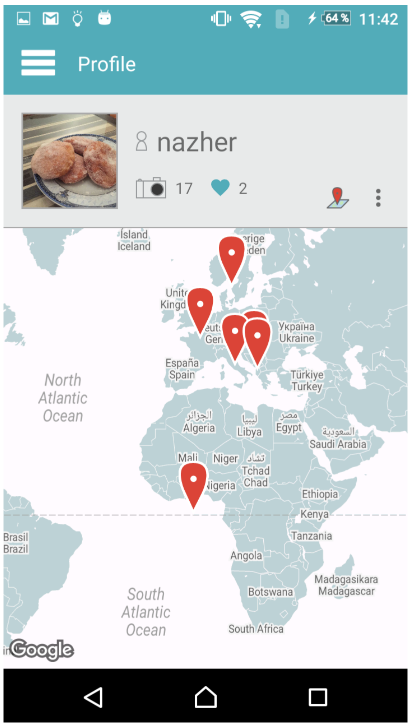
Figure 18 shows the different views of a user's profile. To the left, in Figure 18a, one can see the view that shows when tapping on a user's profile. Here, the user's 360° images are shown on a timeline with rectangles containing thumbnails of the images, the location, date of upload and how many *likes* the image has (the number to the left of the heart). The heart icon can also be pressed to "like" the image or to remove the like if the user has previously liked it. Pressing a thumbnail shows that panorama in the image viewer, as shown in Figure 17a. Pressing the button shaped like a map with a marker on shows the view in Figure 18b, with a map similar to *Explore* showing only the user's images. Above the timeline and map is the information of the user, with a profile picture, name of the user, number of uploaded images (the number to the right of the camera icon) and total number of likes (the number to the right of the heart icon).

4.1.6 List and search of friends

The view in Figure 19a shows a list of the user's friends and is reached by pressing *Friends* in the navigation drawer. This list is divided into sections based on the first letter of the username. Tapping on a friend in the list shows that user's profile. If the user's friend list is too long to fit on the screen, the user will be able to scroll through it to see all friends. When scrolling, an alphabet scroll bar will appear on the right side of the view which the user can use to get to the desired position in the list. Pressing on the button shaped like a magnifying glass in the upper right corner will show the view in 19b, where the user can search for other users and view their profile or send a friend request.

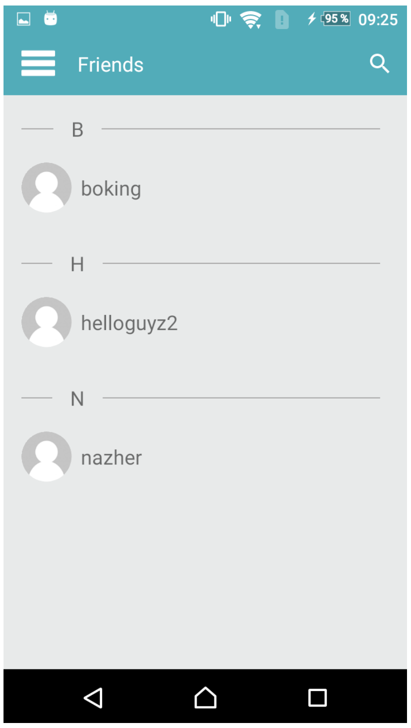


(a) Timeline.

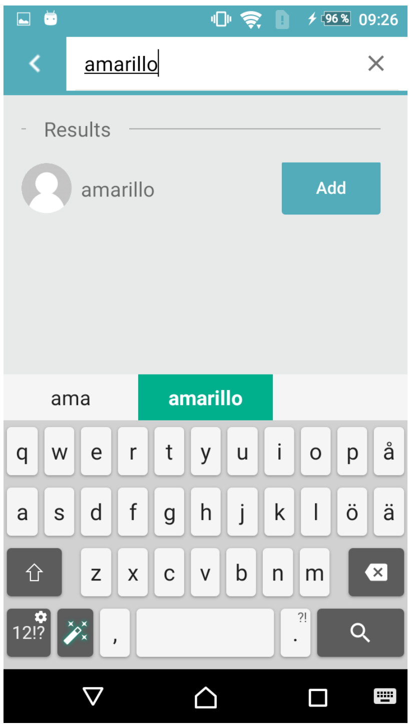


(b) Map.

Figure 18: Figures showing the profile's timeline (left) and map (right) views.

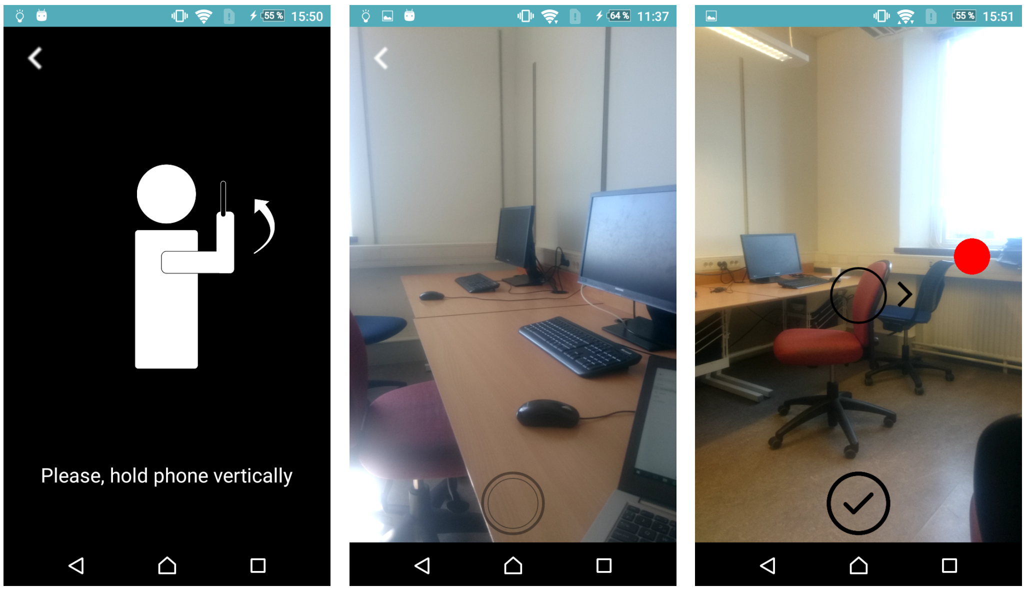


(a) List of friends.



(b) Search for users.

Figure 19: Figures showing the user's list of friends and how to search for users.



(a) If phone not vertical. (b) Before image capturing. (c) Image capture in progress.
Figure 20: Figures showing the view when a user wants to capture a panorama, but is not holding the phone vertically (left), before starting image capturing (middle) and during image capturing (right).

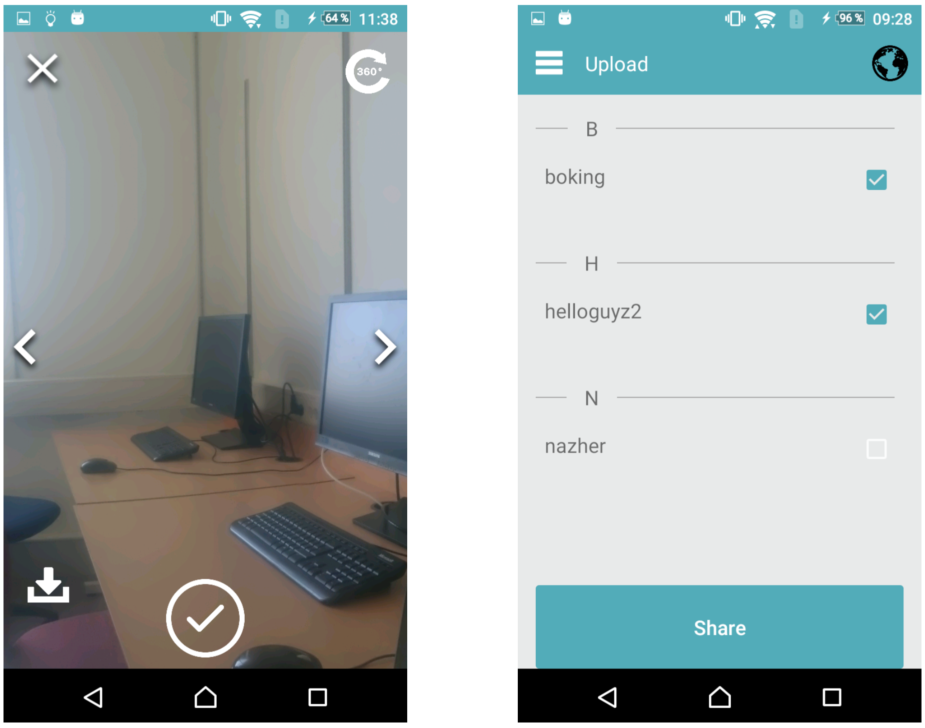
4.1.7 The camera and capture sequence

When the user presses *Camera* in the navigation drawer or the capture button in the *Explore* view, either the view in Figure 20a or in Figure 20b is shown. Which one is shown depends on the rotation of the phone. If the phone is vertical, the preview from the camera will show, as in Figure 20b. If it is not vertical, the view in Figure 20a will show until the user holds the phone vertically. This feedback is presented to ensure that the user keeps the phone vertical throughout the image capturing.

To start the capture process, the user needs to hold the phone vertically and then press the round button placed at the bottom. To capture the next image, the user rotates the phone until the dot is in the middle of the "aim" (black circle) for one second. The arrow beside the aim shows in which direction the dot is. The dot will turn green when it is inside the circle. When the user presses the button again or when the number of images is enough for a 360° panorama, the images will be processed into a panorama. The recently created panorama is then shown in an image viewer, see Figure 21a.

The image viewer in Figure 21a is shown after the user has captured and created a panorama. Here, the user can choose to discard the image by pressing the cross in the left top corner, download it to the phone by tapping the download symbol in the left bottom corner or share it by pressing the button in the middle of the bottom shaped like a check mark. Otherwise, the viewer itself is the same as the one in 17a.

If the user chooses to share the panorama, the app will show the view seen in Figure 21b. Here, the user can choose which friends it wants to share the panorama with by tapping on the checkboxes, making them checked or unchecked. The user can also upload the panorama publicly by tapping the button shaped like a globe in the upper right corner. Then all friends, as well as other users, will be able to see the panorama. However, if we had implemented notifications, only the friends chosen to share with would get a notification.



B

boking ✓

H

helloguyz2 ✓

N

nazher ☐

(a) The recently taken image in an image viewer.

(b) Share view.

Figure 21: Figures showing the image viewer where the user can inspect the recently captured panorama (left) and share view (right).

4.1.8 Local image upload and settings

Lastly, we have *Upload* and *Settings*. When tapping *Upload* from the navigation drawer the application starts a file browser, of the user's choice, to browse the images saved on the phone. Here, the user can choose an image and then upload it to the app. When the image is chosen, the same share view as in Figure 21b will be shown. *Settings* is not implemented at this moment.

4.2 Evaluation of the capture and creation of 360° images in 360World

Table 2: Evaluation of different ways of capturing the images that are to be stitched into panoramas and 360° images.

Way of capturing	**Evaluation of the image stitching**
Capture images by rotating the phone around its own axis of rotation, by having the phone attached to some sort of ”tripod” and rotating the ”tripod”.	This way 360World can capture full 360° images, as the camera is only rotated (no translation) and is steady on a ”tripod”. Example of a panorama captured like this can be seen in Figure 22, which is almost a full 360° image, apart from a missing image at the left part of the image.
Capture images by rotating the phone around the user's axis of rotation, by having the user holding the phone and rotate himself or herself.	This way 360World has not captured any full 360° images, though some very satisfying panoramas. The result depends on how steady the user is and how far away from the body he or she holds the phone. 360World usually skips some pictures or crashes when attempting this.

In Table 2 an evaluation of two different ways of capturing the images to be stitched into a panorama be found. An evaluation of how well the image stitching behaves in different situations, for example bad light conditions, can be found in Table 3.

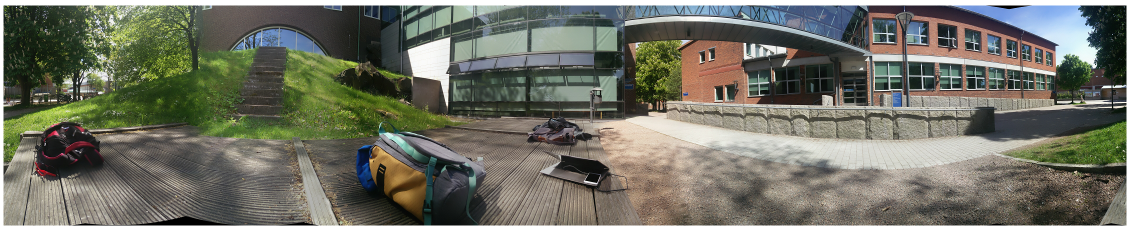


Figure 22: An example of an almost full 360° image, captured using rotation around the axis of rotation of the phone.

Table 3: Evaluation of when 360World can create panoramas and 360° images.

Situation	**Evaluation of the image stitching**
Objects in movement during capture sequence.	This often results in that 360World only includes one of the images with the moving object, and only rarely that it blends the images creating artifacts (as in Figure 25). If the object moves but not in the seams, it can create panoramas with several versions of that object.
Strong light in parts of the captured area.	This is a difficult situation for 360World. It either skips some of the images, and only stitches the images with the same light conditions, or creates a panorama from all images but with a fairly visible seam between the light and dark images.
Area with none or small differences in the captured area, e.g. a white wall.	This is also a difficult situation. When attempting to stitch images captured at this area, 360World usually crashes. This could be because the feature detector cannot find any features in the images.
Bad light conditions.	This is another difficult situation for 360World. When it is too dark, the image stitching does not work at all and the app crashes. Though it is hard to say when it is ”too dark”. In Figure 26 one can see an example of bad light conditions that the image stitching work in. However, it does not work if it would have been darker than this.
”Good” conditions for photography, i.e. not too dark and not too large differences in light.	Here 360World creates very satisfying panoramas with few or no visible seams. Examples of this can be seen in Figure 23 and 24.

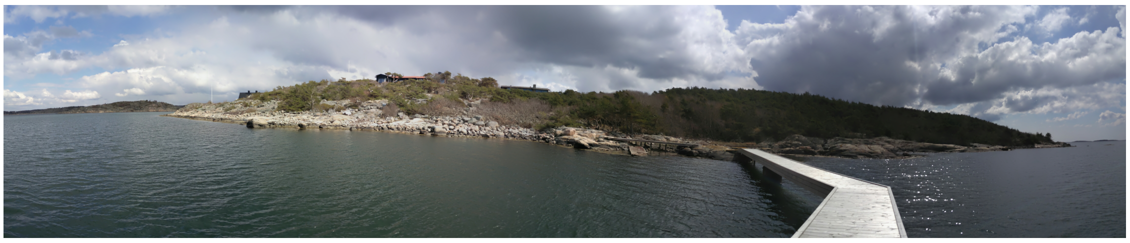


Figure 23: Example of a panorama captured with 360World in ”good” conditions.

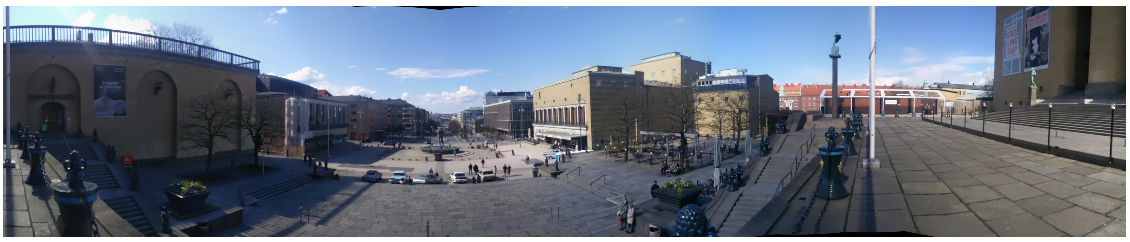


Figure 24: Another example of a panorama captured with 360World in ”good” conditions.

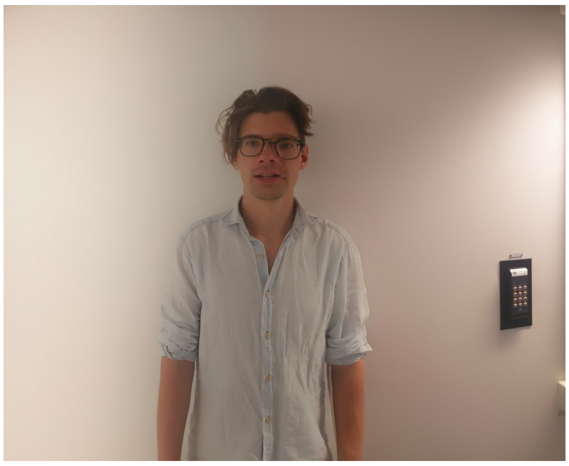


Figure 25: Example of what can happen when an object moves between captured images, in this case 360World stitched the images together creating visible artifacts around the face.

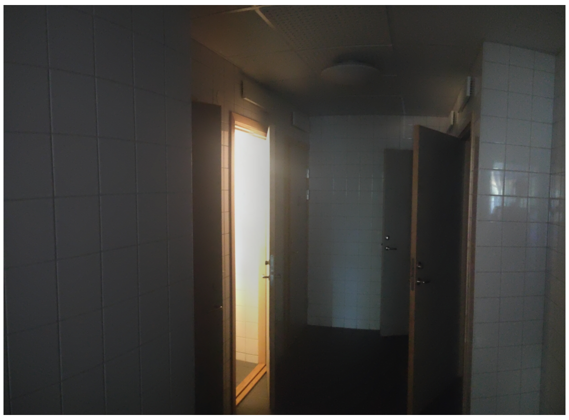


Figure 26: Example of bad light conditions where the image stitching in 360World works.

4.3 Heuristic evaluation of the usability

When the implementation neared its end, the group performed a heuristic evaluation on the entire prototype to determine if our project has reached its purpose. The details of the result can be found in Appendix D, but below is a summary of it.

Overall, the feedback from the system and the user interface is good, but is in some minor cases, missing. The interface itself is clear and easy to navigate, where the objects that are possible to interact with are clearly distinguished from which are not. The language of the application is short and concise without using technical language and complicated words. One important aspect that the application makes sure not to do, is to punish the user for making a mistake. Instead the user can in most cases return to where they were before. Our prototype is consistent when it comes to icons, buttons and the overall design of different screens. In general, the prototype minimises the risk of errors and the amount of things the user has to recall. However, the prototype can in some cases have multiple ways to perform a task, that overall improve the flexibility and the efficiency of use. The application shows only what is really needed by the user with relevant information and action. The documentation that exists is written in plain and simple English with visual aid if the explanation is not understood and error messages use the same idea for the user to recover from these errors. Other than this documentation the application does not require any since the interface can be considered self explanatory.

5 Discussion

In this section, the method and the choices made in the project is discussed, as well as the result. We also discuss what further development of the application can focus on and make a general evaluation of the project as well as discuss the possible impact 360World might have on society.

5.1 Result and method discussion

The following section discusses results and method.

5.1.1 User interface design

We performed the heuristic evaluation ourselves, which introduced potential bias and limits the ability to see the prototype from different points of view. If we had invested in expert evaluators from outside of the project group, it might have been possible that the results covered problems that we did not notice. Another problem that heuristic evaluation might have is that it is possible for the evaluators to miss important usability problems. It is also common that the evaluators present more trivial problems that might not matter much for the end user. From this evaluation, we learned that there are many things we did not think about that actually mattered for the usability of the product. For example, visually showing that something is loading could have been done in more situations.

A way of testing the usability other than the heuristic evaluation would be to let real users from the user base test it. This would give us the perspective of a possible user of the application, who may notice problems that we do not. Having real users evaluating a prototype or product gives a deeper understanding of what they find difficult and what they do not understand. Moreover, it can potentially expose larger problems related to usability within the application. A usability test with real users also highlights how satisfied they are with the application or prototype. Thus, this might give a better understanding of how user-friendly the app is than from a heuristic evaluation but was not done due to the short period of time of the project.

This project, as mentioned before, used Scrum. However, the design process could benefit from the use of another agile development method, for example AgileUX, since it also provides iteration on the design and continuous evaluation of the design, which results in the possibility of a better graphical user interface. We could also benefit from using some kind of design process during our design phase instead of using only personas, an example being Goal Directed Design Process. Something we should have changed after creating hi-fidelity sketches was creating new design prototypes based on the evaluation of previous ones. This would enable us to address more of the issues presented in the heuristic evaluation, since we would have found them earlier in the development process.

5.1.2 Database and servers

An important aspect of the database implementation that we could have done better is the research and planning phase. We assumed that Android devices would be able to connect to database server software directly, much like you can with normal Java programming on a personal computer, but this was not the case. For a number of reasons like security, speed, performance, and accessibility, the approach when querying databases in Android is to use intermediary query scripts on a web service. This had a significant impact on our project time line, because none of us had encountered web server PHP programming before, which meant that a portion of the time resources that were allocated for other purposes had to be used to learn PHP and do the programming. If we had prepared ourselves better, this would likely have been more efficient and as a result, we would have perhaps been able to work on some features that we did not have time for.

The servers were hosted on a machine owned by one of the project group members, which meant that we had greater control of what software was installed and how it was configured. A negative side to this was stability as far as downtime goes; the machine would crash occasionally when the owner was not at home to reset it, which meant some time losses in the programming stage when the group could not connect to the machine.

5.1.3 Map-based image presentation

The explore view and the map-based user profiles are the staple features of 360World, and as such, it was important to get them right. One potential problem relating to the explore view, that would have to be remedied for 360World to be commercially viable, is the downloading of image information from the database. Currently, all available information is downloaded upon initialisation, which would quickly become untenable with a growing user base. Some kind of location based filtering, and perhaps caching of marker information would have to be implemented in order to cope with this. We discussed dealing with this issue during the project, but ultimately decided it was not worth the time investment, as the naive approach of just downloading all the data would be sufficient for the prototype.

Additionally, we ran into a problem with the retrieval of the location at which an image was taken; for some coordinates, such as those corresponding to locations at sea, the geocoder would not always return a valid location, which led to us having to implement an algorithm for finding the closest valid location. This algorithm led to more requests to the geocoder, which meant that we would reach the geocoder API's request limit rapidly, and for this reason, we decided that a search resolution of every 45 degrees would have to suffice. This can easily be changed by adjusting the value with which the angle is incremented. The accuracy of the algorithm could be improved by increasing the search resolution, as it would increase the possibility of finding places near the original location, or by using a different algorithm which searches inside the entire circle. In the case when our current algorithm does find the location from the first given coordinate, but does not find the city that it should show in the profile view, then showing the state instead solves the problem.

5.1.4 Creating a $360°$ image

Our largest difficulty with the image stitching was to understand OpenCV, due to limited documentation being available about the different classes and what algorithms each class had implemented. This made it difficult to understand how they were supposed to be used and we had to do a lot of research on how others have implemented image stitching using OpenCV. If we instead had chosen to use a library that is not free, there would probably been more support available from the developers and clearer instructions on how it was supposed to use.

The fact that we chose to implement two algorithms for image stitching - one simpler to use, one more complex - gave us access to a prototype which could stitch images early on. The implementation of the more complex variant brought a lot of insight into how image stitching works that could not have been gained otherwise, though the quality gain was not as significant as we had hoped. OpenCV's *Stitcher* class seems to be much more robust than our own algorithm, and we did not notice any noteworthy difference in computation time nor image quality.

The final image stitching makes satisfying panoramas if one does not try to make a full $360°$ panorama. With our own algorithm, the stitching often processes for several minutes and then eventually crashes. Using *Stitcher*, we have captured some $360°$ images, but for the most part, it does not include all images and thus creates a panorama that is not fully $360°$. However, when not trying to make a panorama out of all twenty images, both algorithms work fine. We suspect that our own algorithm is not robust enough and that *Stitcher* perhaps needs the images to overlap more. The image quality, however, is surprisingly good, except when trying to combine images with a large difference in exposures.

Further, the implemented algorithm in OpenCV does not consider camera translation between the images, only rotation. But since users of 360World probably will capture their images using a hand-held mobile phone and not on a tripod, there will be some translation between the images. Therefore, for 360World to be able to create full 360° panoramas, we should have implemented an algorithm that also takes translation in consideration, though this would probably have been more complex. For a more user-friendly application, one should also need to decrease the computation time for creating a panorama, as it takes several minutes for the prototype to create a larger panorama.

5.1.5 Capture and display panoramas

At first when trying to implement the sensor functionality, we used the geomagnetic and magnetic field sensors, which in turn use the accelerometer and magnetometer physical sensors, to provide the orientation of the device. This gave us the rotation, but the purpose of this combination of sensors is to provide an accurate direction *in relation to the magnetic field of the earth*; the output we got was not very accurate and would give spikes in the data every now and then. We spent some time trying to make this data more accurate by, for example, implementing a low-pass filter, but then decided to change the sensor type to the rotation vector which worked much better.

As mentioned in the method section, an alternative to guiding the user through the capturing sequence with dots could be to only represent the process with a horizontal line, with an indication of the current direction marked on it. One problem with this is that the user does not get any feedback about the pitch (rotation around x axis) of the device other than whether or not it is within a certain range that allows for capture. If the vertical offset is different in many of the captured images, the stitching algorithm may have trouble to effectively stitch the photos together. We decided to try to improve upon the idea of the line and implemented the dots, which move both vertically and horizontally, providing greater feedback to the user.

Another choice we had to make was the selection of the amount of pictures required in order to capture a full panorama. After some testing, we decided to go with 20 pictures, however, that number could have been both increased and decreased. It is desirable to have an amount of overlap in each pair of pictures, but at the same time, the user should not have to spend more time than necessary to capture a full 360° image. The probability of successfully stitching the panorama might be increased while raising the number of photos needed, but the stitching process will also take longer.

One big problem we had was finding a proper method for implementing the image viewer. This task was underestimated by the group and took more time than we had anticipated. At first, we looked into using the PanoramaGL library for Android, which had a viewer that supported all kinds of panoramic images (spherical, cylindrical, etc.), which would have been better if the app was to support spherical images in the future. The problem is that it only supports images with a max size of 1024x1024 pixels, which is too small for our purpose. After this, we tried several other methods that include using Android ScrollView and ViewPager components, but none of them seemed to support the infinite looping and navigation features. After some time of trial and error, the method that best worked for us, and that we settled on, was the use of a canvas and multiple images to simulate the looping and wrapping.

5.1.6 Evaluation of Scrum

Since it was the first time many members of the group worked according to Scrum, it was difficult to know what preparations we needed to make in the beginning of the project in order to be able to work with Scrum in such a way that it would be helpful to us. If we were to work with Scrum in another project now, it would be much easier to set up and also more helpful during the project. This is why we believe that Scrum, while having a steep learning curve, can be very useful when the basics are grasped.

We only used Trello in the beginning of the project, as it was a very helpful tool when not knowing where to start. Afterwards, when we knew what tasks to next work on more naturally, we did not use Trello as much. Though, we feel now that we would have been helped by using it more, since it would have made it easier to see our progress and what parts we needed to work more on. One thing we should have done better in the beginning of the project is to define more specific use cases, e.g. instead of "implement profile", we should have defined several "implement this function in the profile". This would also have forced us to specify our application more precisely.

The sprint reviews ensured that we were updated about what the rest of the group had done during the last two weeks. This helped us in making well-informed decisions about what parts of the project we should prioritise at the moment, since we knew what the status of the different areas was. Dividing the workload between us was also made easier by the fact that we knew what all group members were working on, and which area that member was specialising themselves in. Moreover, the sprint reviews forced us to continually evaluate our work and project structure.

Though, we could not work fully according to Scrum, since we did not have a product owner. This made us lack the feedback we otherwise would have gotten about the product. Instead, we have only evaluated the product within the group. A product owner would have been a set of fresh eyes, and the question of what functions and features carried the most importance would probably had been better defined.

5.2 360World in society

The 360World prototype has no significant impact on the environment in its current form. However, if it were to be scaled up, with say hundreds of thousands of users and the server infrastructure to accommodate for this, it might leave a more significant footprint. Then a discussion could be held regarding what sort of power, renewable or fossil, is used to drive the servers and under what kind of environmental and social conditions the electrical components are manufactured. In theory, given a hypothetical advancement of mobile phone camera technology, external 360° cameras could be made redundant with an application such as this. This would mean less emissions and waste from manufacturing. The current image quality leaves much to be desired in the context of professional photography, which is why the above is unlikely considering the current state of the art.

If we look at the social and economical aspects, an application such as 360World could expose parts of the world to people who otherwise never would have seen them, because of the locational emphasis of the sharing functionality and the fact that 360° images provide an interesting way to capture the essence of a place. This could potentially make it easier for small businesses in regards to promoting themselves, or increase revenues from tourism. Business such as museums could, using a profile on a hypothetical, scaled up version of 360World, display their exhibitions to a whole new user base.

5.3 Project evaluation

Since we did not have enough time to do everything we wanted to in the beginning of the project, it would have been better to narrow down the scope even more. Now, since we had many different parts (database, server, graphical design, image capturing) the project became broad as well as heavy, and we did not have time to do all of them in a satisfying way. Instead, we could have chosen to focus on only some parts of the app and implement them completely. We could also have connected the app to Facebook so we had not needed to implement a database ourselves. Though, it is worth noting that the project actually was more narrow in the project description, but we felt forced to develop the idea further and do more so the project would be more unique. This also made us lose some weeks in the beginning for brainstorming ideas instead of starting on the project.

We should also have prioritised better in what order we implemented the functions of the app and focused on implementing the most important ones and the ones that made this app unique first. Instead, we spent a lot of time in the beginning on the basic social media functions, like creating an account and logging in, since we thought that we would have enough time to do everything.

Moreover, we spent too much time on discussing small design decisions, since we did not have a product owner that could take those decisions for us. Now, we discussed everything until the group was satisfied which took a lot of time that we could have spent better. We could have, for example, "hired" someone to be our product owner or chosen one in the group to take all decisions about design and functions. This problem arose from that the students who came up with this idea were not part of the project. The project would probably have been clearer if those students had explained the idea to us before we started with the project and had them as a product owner.

Overall, the group is satisfied with the project and the resulting prototype, since the app and many of its functions are working well. We have also learnt a lot about how it is to work with a project and a group as large as this.

5.4 Continued development

Since the result of this project only is a prototype, there are a lot of room for continued development, not only improvements of existing functions but also functions that could make the app more unique. The first thing we would do if we were to continue with the project would be to do proper user testing and evaluation of the app, so we would know what functions the users are missing and what the continued development should focus on. Already from the beginning of the project, we also had a list of limitations that would not be addressed in this project, see Section 1.2. Most of these would be needed to be approached to reach a complete application. The app would, for example, need to support more devices and the storing of data on the server should be encrypted due to security issues before we launched the application on the market. Also, the notifications view and push notifications as well as the settings view would need to be fully implemented.

5.4.1 Map-based image representation

We have two different ideas of how to make the map view more unique. The idea we had from the beginning was that only recently captured images were to be shown on the map, so the images would show how different places look like right now. But all images from a user would still be available on the user's profile. The other idea is to add a function to filter the map after time, so the user could be able to see images uploaded on the same place but at different seasons and years. Both these approaches would make this app more distinctive from Google Street View.

5.4.2 Creating and displaying a panorama

Furthermore, the camera and capture sequence can be improved. The first step would be to review if there is another algorithm or library we can use for image stitching that is more suited for small devices and full 360° images, as the image stitching now takes several minutes and rarely creates the whole panorama. This algorithm should probably also take translation, and not only rotation, into account. One could also experiment more with how many images that is needed for the panorama, if the reason for the algorithm not creating full panoramas is that the images are not overlapping enough.

A natural step would then be to implement so the app also can create spherical 360° panoramas, which also was something we mentioned in Section 1.2 that we would not do in this project. In order to do this, the capture sequence needs to have support for capturing images in more than one direction, but the stitching algorithm should be able to function as well as for cylindrical panoramas. The image viewer, however, would need to be changed to be able to show spherical panoramas.

Another function that can be implemented is so the panorama is stitched during the capture sequence, i.e. so the most recently captured image is stitched together with the already existing panorama for each image. Then the part of the panorama that is done can be shown during the capture sequence, so the user can see how the panorama turns out earlier. Another feature that can be added is to draw and write on the image, so the user can point out important parts of their panorama.

Other popular features are description of the image (which there is support for in the database), comments to the panoramas, tagging the images and adding filters to the panoramas. It is also common for other apps that can create 360° images to have support for connecting an external 360° camera and take pictures to the app with it. To develop the possibilities to explore new places and panoramas, one could implement a feature that views a random image if the user shakes his or her phone. Further, it could also be interesting to be able to view the panoramas with Google Cardboard and other similar VR-headsets.

5.4.3 General improvements

Lastly, we present some general improvements of the app. To start with, one important improvement is to solve all memory leaks and make uploading and downloading from the server faster, making in general a more bug-free and faster app. To make the app more user-friendly one could add a tutorial about the app that should show when the user opens the app for the first time. Before releasing the app to the market it would also be needed to add ways to report abusive or bad images as well as rules about what images that are allowed and not. There are of course also small details that can be changed or added, for example to be able to change the viewing permissions of an already uploaded photo. However, the user evaluation should decide which ones to mainly focus on. Another popular option in today's apps is to be able to create an account using an already existing Facebook account.

6 Conclusions

The purpose of this project was to develop a prototype of a social media platform through an Android application with which the user can create and, with the locational aspect emphasized, share 360° images with other users. We have designed a user interface for the application, implemented the creation and viewing of 360° images, implemented social media features in the form of friends, profiles (both conventional profiles and a mode where the user profile is represented with a map), and image likes. Additionally, we have designed and implemented a database which supports image information storage as well as the mentioned social media features, set up servers for storage of information and images, and implemented an interface between these and the Android application.

The project has been a success in the sense that we managed to implement and integrate the functionality outlined in Section 1.1. However, for potential commercial viability, there are several issues that would need to be addressed; the most significant is perhaps the fact that there are problems remaining when stitching the images, where failure to produce a coherent 360° image often occurs. We can also not fully conclude if the application is user-friendly, since we were not able to test it to the required extent. As far as the locational aspect of the image sharing goes, we are pleased with the look and functionality of the map views, and we feel that they successfully tap into the natural synergy between 360° pictures and locations, outlined in Section 1.

When it comes to answering the question of whether the results constitute a meaningful addition to the social media industry or not, discussed in Section 1, we can only speculate. 360World does provide the user with a lower barrier of entry system for sharing 360° images than currently exists, in our opinion, but the inherent problem with the tediousness of capturing of 360° images yet remains. Whether the synergy between the location aspect of panoramas, and the geotagging we make use of in the explore view, elevate the user experience enough to justify the application in its totality is ultimately subjective, and unfortunately we have only our own opinions to answer this question with. If greater testing opportunities had been available, we might have been able to make a more compelling case for the usefulness of the application.

The project has given us insight into the difficulties associated with planning and executing a project of this size. We have also learned a lot, specifically about working according to Scrum, and its advantages. We have also gained insights into the Android application development process, algorithms for image stitching, database design, setting up servers, as well as the integration of all these parts into one system.

This report has described the process of developing the application 360World, and the result of our work, as well as what can be further developed if one were to continue the project.

References

[1] J. I. Cole, "The world internet project international report - seventh edition," 2017, Visited 2017-01-30.

[2] ——, *The world internet project international report - fourth edition*, `http://www.digitalcenter.org/wp-content/uploads/2013/01/2012wip_report4th_ed.pdf`, Visited 2017-01-30, 2012.

[3] ——, *The world internet project international report - fifth edition*, `http://www.digitalcenter.org/wp-content/uploads/2013/12/2013worldinternetreport.pdf`, Visited 2017-02-05, 2013.

[4] A. Hornung, *Space needle 360 panorama*, `https://commons.wikimedia.org/wiki/File%3ASpace_Needle_360_Panorama.jpg`, Downloaded 2017-02-09.

[5] Ximeg, *Spherical panorama-226-nürnberg 2013pano3*, `https://commons.wikimedia.org/wiki/File%3ASpherical_panorama-226-N%C3%BCrnberg_2013pano3.jpg`, Downloaded 2017-02-09.

[6] (n.d). Application fundamentals, Android Developers, [Online]. Available: `https://developer.android.com/guide/components/fundamentals.html` (visited on 2017-03-11).

[7] (n.d). Activities, [Online]. Available: `https://developer.android.com/guide/components/activities/index.html` (visited on 2017-03-11).

[8] (n.d). Activity lifecycle. 2017-04-05, Android Developers, [Online]. Available: `https://developer.android.com/reference/android/app/Activity.html#ActivityLifecycle`.

[9] (n.d). Fragments, Android Developers, [Online]. Available: `https://developer.android.com/guide/components/fragments.html` (visited on 2017-04-09).

[10] (n.d). Glossary, U.S. Department of Health & Human Services, [Online]. Available: `https://www.usability.gov/what-and-why/glossary/i/index.html` (visited on 2017-03-24).

[11] A. Cooper, R. Reimann, D. Cronin, and C. Noessel, *About face: The essentials of interaction design*, 4. 2014, p. 11.

[12] (n.d). Personas, U.S. Department of Health & Human Services, [Online]. Available: `https://www.usability.gov/how-to-and-tools/methods/personas.html` (visited on 2017-02-02).

[13] (n.d). Glossary: Usability, Usability First, [Online]. Available: `http://www.usabilityfirst.com/glossary/usability/` (visited on 2017-02-09).

[14] (n.d). Complete beginner's guide to interaction design, UX Booth, [Online]. Available: `http://www.uxbooth.com/articles/complete-beginners-guide-to-interaction-design/` (visited on 2017-02-02).

[15] J. Tidwell, *Designing interfaces: Patterns for effective interaction design*, 2. 2010.

[16] J. Nielsen, "Enhancing the explanatory power of usability heuristics," *CHI '94 Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 152–158, 1994.

[17] W. Garcia-Molina Ullman, *Database systems: The complete book, second edition.* Pearson, 2014.

[18] V. Kumar, "Review of database and prominent programmes," *International Journal of Research in Science And Technology*, 2012.

[19] (n.d). Apple presents iphone 4, Apple Inc, [Online]. Available: `http://www.apple.com/pr/library/2010/06/07Apple-Presents-iPhone-4.html` (visited on 2017-04-08).

[20] (n.d). Android sensor framework, Google Inc, [Online]. Available: `https://developer.android.com/guide/topics/sensors/sensors_overview.html` (visited on 2017-04-26).

[21] U. S. A. Rodriguez, "Indoor positioning using sensor-fusion in android devices," Sep. 2011. [Online]. Available: `http://www.diva-portal.org/smash/get/diva2:475619/FULLTEXT02.pdf`.

[22] R. Szeliski, "Image alignment and stitching: A tutorial," *Found. Trends. Comput. Graph. Vis.*, vol. 2, no. 1, pp. 1–104, Jan. 2006, ISSN: 1572-2740. DOI: `10.1561/0600000009`.

[23] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *Int. J. Comput. Vision*, vol. 74, no. 1, pp. 59–73, Aug. 2007, ISSN: 0920-5691. DOI: `10.1007/s11263-006-0002-3`.

[24] K. Lin, N. Jiang, L.-F. Cheong, M. Do, and J. Lu, “Seagull: Seam-guided local alignment for parallax-tolerant image stitching,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.

[25] C. Liu, J. Yuen, and A. Torralba, “Sift flow: Dense correspondence across scenes and its applications,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 5, pp. 978–994, May 2011, ISSN: 0162-8828. DOI: 10.1109/TPAMI.2010.147.

[26] (n.d). Images stitching, OpenCV, [Online]. Available: http://docs.opencv.org/3.0.0/d1/d46/group__stitching.html (visited on 2017-04-01).

[27] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *Int. J. Comput. Vision*, vol. 60, no. 2, pp. 91–110, Nov. 2004, ISSN: 0920-5691. DOI: 10.1023/B:VISI.0000029664.99615.94.

[28] T. T. Herbert Bay and L. V. Gool, “Surf: Speeded up robust features,” 2006, Read 2017-05-02.

[29] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: An efficient alternative to sift or surf,” ICCV ’11, pp. 2564–2571, 2011. DOI: 10.1109/ICCV.2011.6126544.

[30] (n.d). Orb (oriented fast and rotated brief), OpenCV, [Online]. Available: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html#orb (visited on 2017-05-02).

[31] F. M. Schaukowitsch. (n.d). Pano.net: An interactive application for camera calibration, image stitching and projective operations, [Online]. Available: https://www.cg.tuwien.ac.at/research/publications/2013/schaukowitsch-2013-fls/schaukowitsch-2013-fls-Thesis.pdf (visited on 2017-05-03).

[32] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, no. 6, pp. 381–395, Jun. 1981, ISSN: 0001-0782. DOI: 10.1145/358669.358692.

[33] E. Dubrofsky, “Homography estimation,” Master’s thesis, The University of British Columbia, Mar. 2009.

[34] B. Summa, J. Tierny, and V. Pascucci, “Panorama weaving: Fast and flexible seam processing,” *ACM Trans. Graph.*, vol. 31, no. 4, 83:1–83:11, Jul. 2012, ISSN: 0730-0301. DOI: 10.1145/2185520.2185579.

[35] M. Uyttendaele, A. Eden, and R. Szeliski. (n.d). Eliminating ghosting and exposure artifacts in image mosaics, [Online]. Available: https://ai2-s2-pdfs.s3.amazonaws.com/0b5a/3f27c44bf07df987500b73f09200dd21347b.pdf (visited on 2017-05-05).

[36] P. J. Burt and E. H. Adelson, “A multiresolution spline with application to image mosaics,” *ACM Trans. Graph.*, vol. 2, no. 4, pp. 217–236, Oct. 1983, ISSN: 0730-0301. DOI: 10.1145/245.247.

[37] (n.d). Android studio, Google Inc, [Online]. Available: https://developer.android.com/studio/index.html (visited on 2017-02-01).

[38] (n.d). Github, [Online]. Available: https://github.com/ (visited on 2017-02-01).

[39] (n.d). Learn about scrum, Scrum Alliance, [Online]. Available: https://www.scrumalliance.org/why-scrum (visited on 2017-02-01).

[40] (n.d). Trello, [Online]. Available: https://trello.com/ (visited on 2017-02-01).

[41] (n.d). What’s the difference? agile vs scrum vs waterfall vs kanban, Smartsheet Inc, [Online]. Available: https://www.smartsheet.com/agile-vs-scrum-vs-waterfall-vs-kanban (visited on 2017-02-05).

[42] Lakeworks, *Scrum process*, https://commons.wikimedia.org/wiki/File:Scrum_process.svg, Downloaded 2017-05-12.

[43] M. Brian, *Google’s new ’material design’ ui coming to android, chrome os and the web*, n.d. [Online]. Available: https://www.engadget.com/2014/06/25/googles-new-design-language-is-called-material-design/ (visited on 2017-02-01).

[44] (n.d). Popularity ranking of relational dbms, DB-Engines, [Online]. Available: https://db-engines.com/en/ranking/relational+dbms (visited on 2017-05-02).

[45] (n.d). Google maps, Google Inc, [Online]. Available: https://developers.google.com/maps/ (visited on 2017-02-10).

[46] *Solution guide iii-c - 3d vision*, Halcon, 2016. [Online]. Available: http://www.mvtec.com/fileadmin/Redaktion/mvtec.com/documentation/halcon/halcon-13.0-solution-guide-iii-c-3d-vision.pdf (visited on 2017-05-24).

[47] (n.d). Registration, Matrox Imaging, [Online]. Available: http://www.matrox.com/imaging/en/products/software/mil/tools/registration/ (visited on 2017-05-24).

[48] (n.d). Opencv, Intel Corporation, [Online]. Available: http://opencv.org/ (visited on 2017-02-10).

[49] S. Lee and J. W. Jeon, "Evaluating performance of android platform using native c for embedded systems," pp. 1160–1163, Oct. 2010. DOI: 10.1109/ICCAS.2010.5669738.

[50] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following," *Computer Vision, Graphics, and Image Processing*, vol. 30, no. 1, pp. 32–46, Apr. 1985.

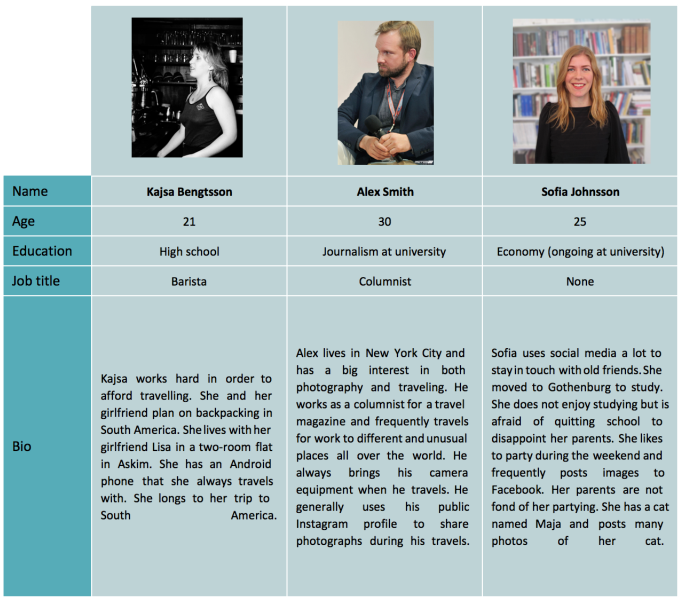
[51] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *The Canadian Cartographer*, vol. 10, no. 2, pp. 112–122, 1973. DOI: 10.3138/FM57-6770-U75U-7727.

[52] Sten. (n.d). The pretty barista, [Online]. Available: https://www.flickr.com/photos/fromtheid/2526258975/ (visited on 2017-05-12).

[53] MotorBlog. (n.d). Journalist mathias becker, [Online]. Available: https://www.flickr.com/photos/motorblog/10401947274 (visited on 2017-05-12).

[54] FORES. (n.d). Malin niemi, [Online]. Available: https://www.flickr.com/photos/fores_se/8651437223 (visited on 2017-05-12).

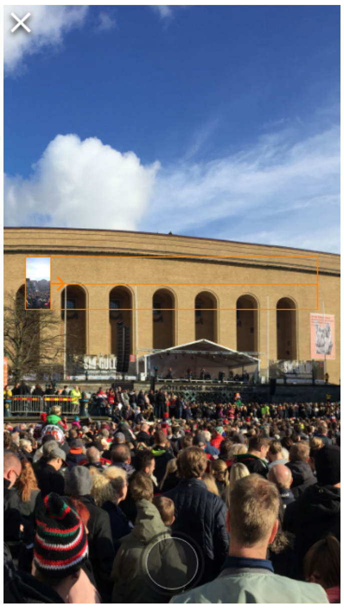
A Personas



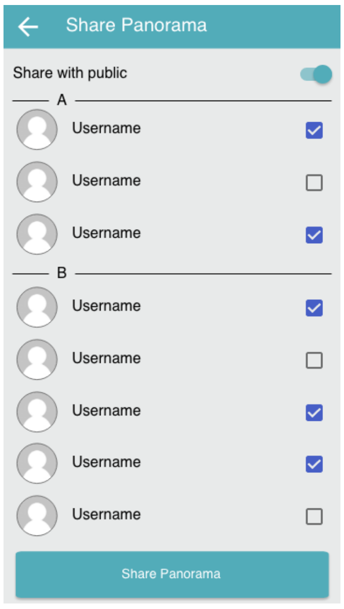
Name	**Kajsa Bengtsson**	**Alex Smith**	**Sofia Johnsson**
Age	21	30	25
Education	High school	Journalism at university	Economy (ongoing at university)
Job title	Barista	Columnist	None
Bio	Kajsa works hard in order to afford travelling. She and her girlfriend plan on backpacking in South America. She lives with her girlfriend Lisa in a two-room flat in Askim. She has an Android phone that she always travels with. She longs to her trip to South America.	Alex lives in New York City and has a big interest in both photography and traveling. He works as a columnist for a travel magazine and frequently travels for work to different and unusual places all over the world. He always brings his camera equipment when he travels. He generally uses his public Instagram profile to share photographs during his travels.	Sofia uses social media a lot to stay in touch with old friends. She moved to Gothenburg to study. She does not enjoy studying but is afraid of quitting school to disappoint her parents. She likes to party during the weekend and frequently posts images to Facebook. Her parents are not fond of her partying. She has a cat named Maja and posts many photos of her cat.

Figure 27: Illustration of the personas that were used during the prototype development. Image sources from left to right: [52], CC BY; [53], CC BY; [54], CC BY.

B High fidelity sketches

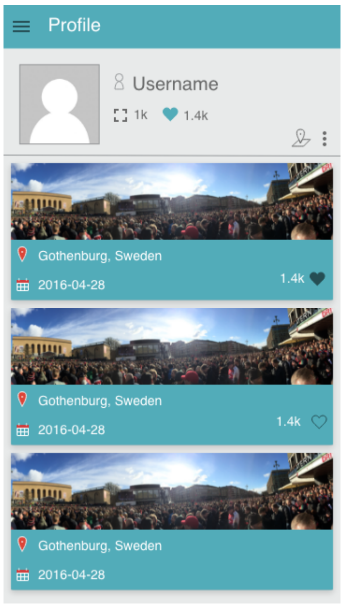


(a) During the process of taking a panorama image.

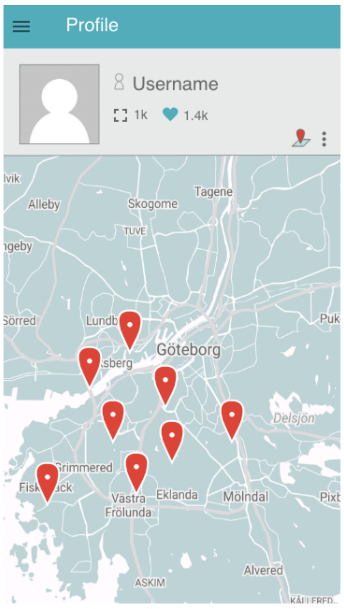


(b) Selecting who should be able to view and who should be notified of the panorama.

Figure 28: Hi-fidelity sketches of the process of taking and sharing a panorama.

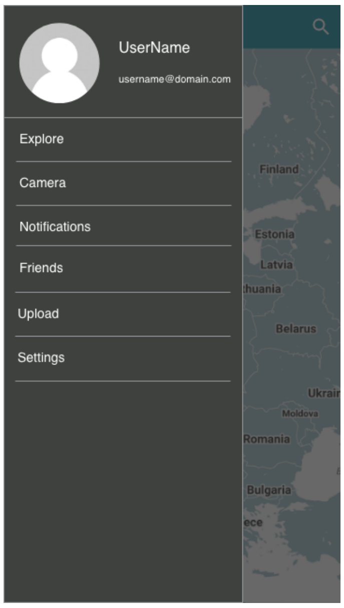


(a) Profile in list mode

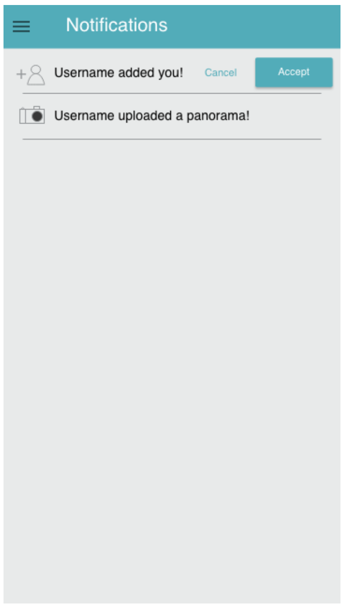


(b) Profile in map mode

Figure 29: Hi-fidelity sketches of the two modes a profile can be viewed in.

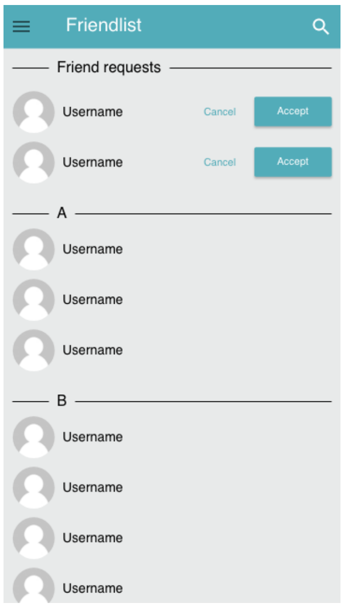


(a) Displaying the navigation.

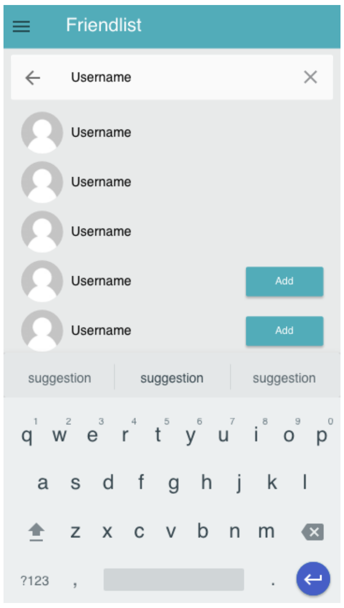


(b) Displaying the different notifications.

Figure 30: Hi-fidelity sketches of the navigation drawer and the notifications screen.

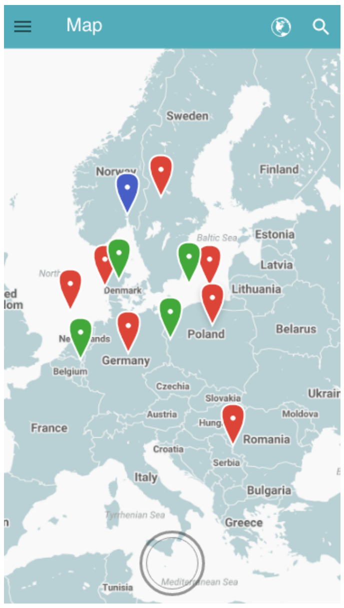


(a) The list of friends.

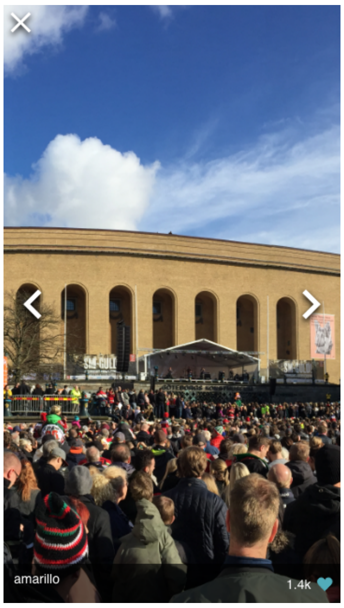


(b) Searching for friends in the list of friends.

Figure 31: Hi-fidelity sketches of the list of friends and the search functionality that exists in this list.



(a) Displaying the explore screen.



(b) Displaying the image viewer.

Figure 32: Hi-fidelity sketches of the 'Explore' screen and the image viewer where the user can view a panorama.

C Entity relationship diagram

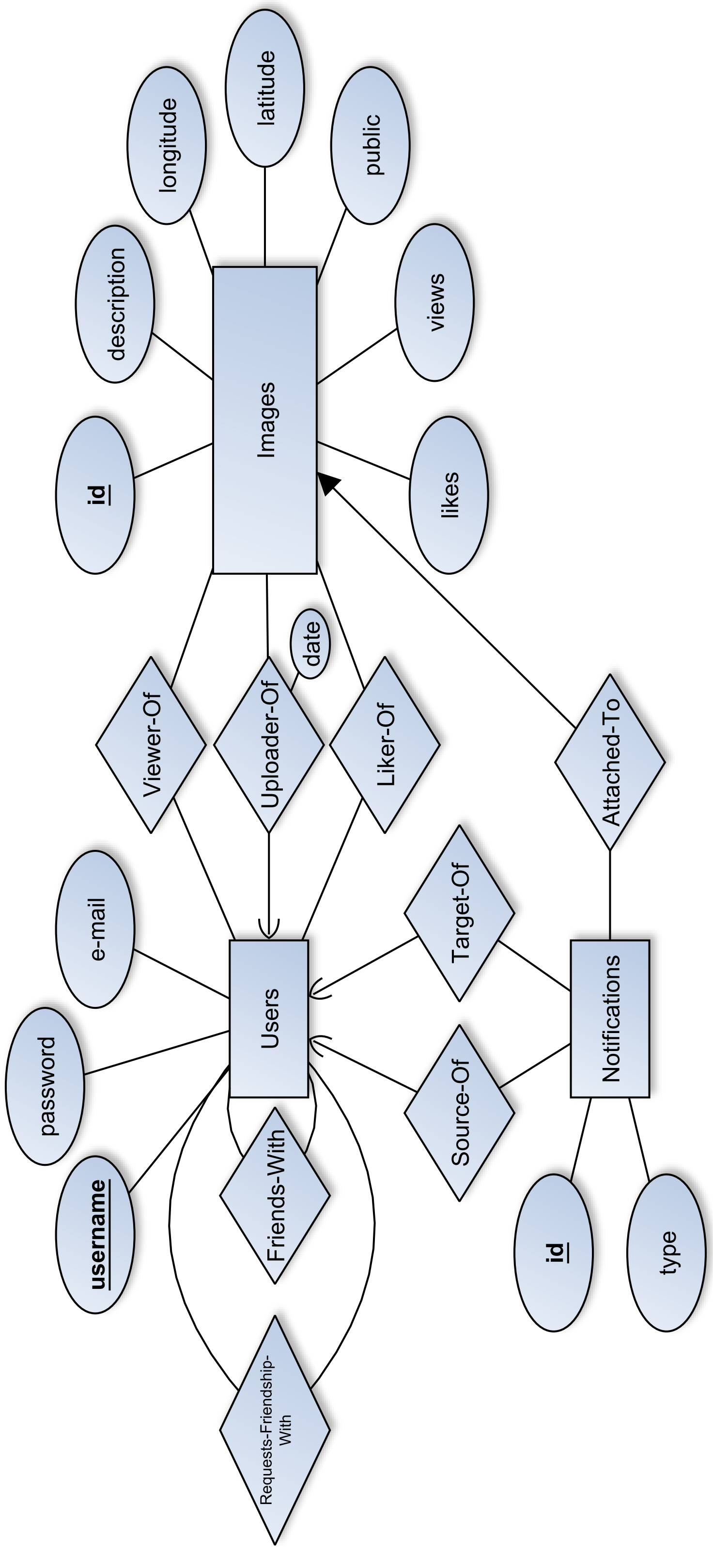


Figure 33: The entity relationship diagram for the database of 360World.

D Results from heuristic evaluation

1. **Visibility of system status**:
   Overall the user interface is clear, it does not have too many interactive objects and the user easily noticed which of the icons and text elements that appear in the background or foreground. For a novice user, in the *Explore* view, the "globe"-icon that changes whether the user is viewing public or friends' images, might be hard to understand, while it for an expert user it might be more obvious.

   Sometimes there is a lack of feedback, for example when loading, or when a profile with no uploaded images is visited. Then it is unclear if the user has any images or not. There exists feedback for this but it is not sufficiently prominent. When it comes to the panorama and camera features, the application clearly states when it is working in the background, which is only done when it is stitching the panorama.

   The map that is used throughout the application has a nice contrast between different objects on the map and the interface. This is also true for the contrast between the map and the toolbar. The reduction in text and objects on the map makes it easier to navigate for the purpose of the application.

2. **Match between system and the real world**:
   The system does not use technical language in texts and notifications, but rather uses terms that are as simple and short as possible. All locations are translated to cities, regions or countries, depending on what the system can find, rather than showing the longitude and latitude of where the photo was taken. It will show "Unknown location" if the location can not be found. However, if the region or city cannot be found while the country is found, it will show "Unknown city" instead of the city or region.

3. **User control and freedom**:
   There are some situations, where backwards navigation is important, for example, if the user is looking at his or her list of friends and selects one of them to visit their profile. Then, there should be a button that takes the user back to the list of friends again.

   In the camera functionality, it is always possible to move a step backwards. However, if the user commits an error when taking individual images for a panorama, it is not possible to undo an individual image. In that case, the user is punished by having to retake the entire panorama and wait for the stitching process before being able to return to the step before taking the images. Other than the above mentioned spots of bother, there is no real punishment for the user when he or she presses the wrong button.

4. **Consistency and standards**:
   Overall the application is consistent with the meaning of icons and buttons, even though some icons might be more difficult to interpret. The application can be very simple to navigate and understand because many of the elements are similar to what can be found in other applications.

   The search button is very clear in what it means and its functionality follows how the rest of the industry uses similar icons. When it comes to the camera and sharing functionality the application can also be considered consistent and it also reuses the layout of the *Friends* view when selecting who should receive the image after sharing it. The application is also consistent when it comes to reusing the same map in *Explore* and in the map view of a profile.

5. **Error prevention**:
   Since the system only takes an image when the camera is aimed at the correct spot, it minimises the risk of error in the stitching process, which in turn makes it harder for the user to commit an error. Other than this mentioned example, there does not exist any implementation of error prevention.

6. **Recognition rather than recall**:
   Overall, there is not much for the user to actually recall and remember other than what panorama they pressed in the *Upload* view when they are on the *Share* screen. When the user is taking a panorama the system keeps track of how many degrees of the image that has already been captured and the user only needs to think about where the next image should be captured.

7. **Flexibility and efficiency of use**:
   The application is not so complicated that expert functions are necessary. From almost every screen the user can reach the navigation and there are no real hidden features, except that it is possible for the user to swipe from the left side of the screen in order to open the navigation drawer.

   A user will be efficient performing the tasks that the application was designed for since it is easy to navigate and use. The only possible place where there are multiple ways of completing a task is when viewing panoramas since it contains three different ways of navigating and viewing the panorama.

   On the profile view, it might be more complicated to use the map to view the panoramas instead of the list, this is plain flexibility for the user to choose how he or she wants to view a profile. On the map view, there is room for improving the efficiency of use since it does not move the screen to cover the locations of the profiles uploaded panoramas and it therefore requires the user to navigate the map to find the panoramas that the profile have uploaded. When it comes to the camera functionality, it does not provide multiple ways of use.

8. **Aesthetic and minimalist design**:
   Overall the application shows only what is really needed in order for the user to perform the tasks, while menus are context sensitive so it only show the information and actions that are needed. When it comes to the *Log in* screen, it does not show any way for the user to retrieve his or her password if he or she has forgotten it. This is not a good practice since this hides relevant information and functionality from the user. Some might think that the extra text ("Minimum of five characters") on the *Create account* screen might be irrelevant or rarely needed, but it can also be seen as vitally important in completing the create account process.

9. **Help users recognise, diagnose, and recover from errors**:
   There does not exist any visual error codes in the application, which is very good since they are hard for non-developers of the application to understand and most users might not know how to use them in order to find a solution. The errors in the application does not require direct interaction, but rather notifies the user that something has gone wrong and might require their attention. The notifications are written in as simple English as possible while not containing any technical language and propose what the user can do to solve the problem. The information that is presented when the phone is not held vertically (see Figure 21b), is presented in a meaningful way for the user since it uses a clear and not too detailed image while also showing a short sentence in a simple language explaining the problem and how it is solved.

10. **Help and documentation**:
    Overall the application does not require documentation to perform the different tasks that it was designed for, but there are instructions when it comes to taking the panorama and holding the phone correctly since this is vitally important for the functionality of the application to work. While taking the panorama there are also visual guides to show how the user should move the device. Many of the other tasks do not require documentation since they are fairly self explanatory and simple.