Mobile Application Development with Swift

DTU - Technical University of Denmark

NeighBird

Rasmus Gundel - S153980
Sara Nordberg - S150159

20. april 2018

Indhold

1 Introduction

This report contains information about an iOS app developed in the course *Mobile Application Development with Swift*. It will describe the app and its purpose. Also, it talks about the requirements for the app and the analysis behind the work. Later an overview of the design of the app will be shown, alongside with some selected implementations. At last it will talk about further improvements.

1.1 NeighBird

The app we have developed is called *NeighBird*. It is an app that will act as a neighbourhood watch. The app can help you keep an eye on your home - by the help of your neighbours - when you, yourself, is not able to.

The idea is that you have some sort of security alarm set up at home, e.g. an alarm or a camera. Then you sign up to NeighBirds and is attached to certain groups within the app. An example could be a group that consists of the different people who live on the same street as you do. Then when you get an alarm from your security system you can alert your NeighBirds. This is done by choosing a predefined message and the group who will get the alert, and then simply just swiping the alert button. Then all the people in the specific group will get your alert and can then answer back if they are able to help check your house.

In the long run the app should be able to connect to your home security system so you only need to go one place, the NeighBird app, when you are in need of help.

The app has been developed in collaboration with Benjamin Bidstrup who is a graphic designer and Michael Jacobsen who is a key account manager. They approached us with the idea for NeighBird and we have been in close communication with them throughout the development process.

2 Analysis

2.1 Functional Requirements

- Personal user account
- Creation of groups consisting of users
- Able to send chat messages within a group
- Quick and easy alert function

2.2 Non-functional Requirements

- The application must be developed using Swift
- Language most be Danish
- Follow Apple Design guidelines
- User friendly interface
- The app should be free

The requirements are set based on the feedback from Benjamin and Michael. They wanted a simple and intuitive neighborhood watch app which could compete with products already on the market but without any expensive subscription fees. Further more they wanted the user to be able to use any security system on the market.

2.3 Use Case Descriptions

We have created use case descriptions based on what we think the flow of the app should be. These functions are what should be implemented in the app as a bare minimum. They can be described as the criteria for what our minimal viable product should be able to do. We will examine four different use cases; Create user, log in, create group, and send alert.
The first use case - user creation - is a basic requirement for many apps that needs personal information, and the first thing which we need for our app to work.

2.3.1 Create new user use case

Use case: Create new user
ID: C01
Brief description: Create a user in NeighBird
Primary actors: The user
Secondary actors: None
Preconditions: The user has the app installed and an internet connection
Main flow 1. The user opens the app 2. The user clicks "Ny Bruger" 3. The user inputs his personal information 4. The user clicks "Opret Bruger"
Postconditions: The user is created and a confirmation email is sent to the email specified.
Alternative flow: 1. The user has used an email already in use 2. An error message is shown

This use case description is the first action any user will have to do before being able to further use the app. The workflow starts with the user opening the app and pressing the "Ny Bruger"button. This will take the user to a screen where they will be able to type in their personal information and choose a picture if they desire. They will then use a button and their user will be created. If the user inputs an email already in use an error will be shown. If the user at any point decides not to create a user they can click the "Fortryd"button which will take them back to the login screen.
The outcome from this use case should be that a new user has been created.

2.3.2 Login use case

Use case: Log in
ID: C02
Brief description: Log in to NeighBird
Primary actors: The user
Secondary actors: None
Preconditions: The user has the app installed and an internet connection
Main flow 1. The user opens the app 2. The user inputs email and password 3. The user clicks "Login"
Postconditions: The user is logged into NeighBird
Alternative flow: 1. The user wrote their email or password wrong 2. An error message is shown

When the user has created their own personal account they will be able to log in from the log in page. This is easily done by typing in their email address and their personal password. If email and password match the user will be logged into NeighBird, and will be able to use all of its functions. In the case that either email address or password is wrong an error message will be shown, and the user will not be able to log in.

2.3.3 Create new group use case

Use case: Create new group
ID: C03
Brief description: Create a new group
Primary actors: The user
Secondary actors: None
Preconditions: The user has the app installed and an internet connection
Main flow 1. The user navigates to the group panel 2. The user clicks "Tilføj Gruppe" 3. The user inputs a name for the group 4. The user clicks "Tilføj Medlemmer" 5. The user searches and selects for people that they wish to invite to the group 6. The user clicks "Opret Gruppe"
Postconditions: The user has created a new group by the name selected and other users have received an invitation
Alternative flow: 1. The user clicks "Fortryd" 2. The workflow is stopped 3. The user is navigated back to their group panel

When the user is logged in they can navigate to the group page. In here they will be able to create a new group by clicking an add button. They will then be taken to a page where they can type the name of the new group and select which users should be included. There is no limit to the amount of users that can be in group. If the user decides on not wanting to create a group they can simply click the "Fortryd"button which will take them back to the previous view.
The outcome of this use case is that a user has created a group and is the owner of the given group. Other attached users will have a new group in their account.

2.3.4 Send out an alert/chat message use case

Use case: Send out an alert
ID: C04
Brief description: Send out an alert to a specific group
Primary actors: The user
Secondary actors: None
Preconditions: The user has the app installed and an internet connection
Main flow 1. The user opens the app 2. The user navigates to the alert panel 3. The user writes an alert message 4. The user selects which group they want to alert, from at dropdown menu 5. The user swips the "Alarmér NeighBirds"button
Postconditions: The users in the specified group receives a push notification with the specified message
Alternative flow: 1. The user clicks the "tilføj billede"button 2. The user selects the appropriate image 3. The user writes an alert message 4. The user selects which group they want to alert, from at dropdown menu 5. The user swips the "Alarmér NeighBirds"

When a user is a member of a group they are able to send an alert. This is done by navigating to the alert page. The alert page will also be the initial page you see after you have logged in. Here they will be able to select the message that should be sent out with the alert and then select which group should be alerted. When the user has selected both message and group they can send the alert using a button. If one of the two is not selected an error will be shown, and the alert will not be sent.
The outcome of this use case is that a group has been sent an alert from a user within the alerted group.

2.4 Core Function

Based on the requirements and our use cases we have determined what our core functions are. A user should be able to send out a message asking for help to a group. Each group should consist of people who have been invited by the owner of the group. Each user should be able to be a member of multiple groups and be able to send an alert or respond to one for any of these groups. An alert is a message which is of high priority and a notification should be pushed whenever an alert is sent. All this should be build in a intuitive user interface.

3 Design

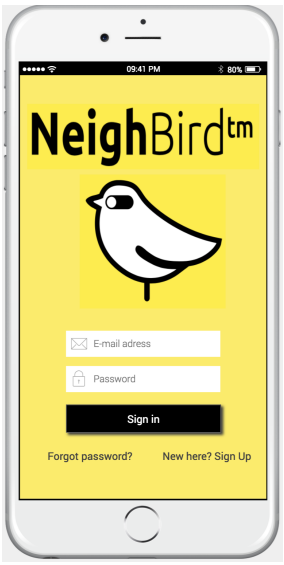
3.1 UX Design

The design of the app should be very simple and easy to navigate. Since it is an app to send out serious alerts to get help, it needs to be very quick and easy to do so. From the very beginning the app has had the name NeighBird, and the logo to go with it has been the white bird with a swipe button for an eye. Together with the iconic bird the color yellow and black is used as it signals danger.

3.1.1 Iterations of design

This section will talk about the different versions of the UX design. We started out by making prototypes of the app using *JustInMind*. In the following we will see the evolution of the design.

(a) Version 1

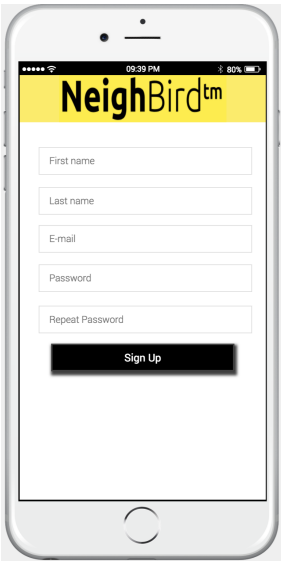


(b) Version 2

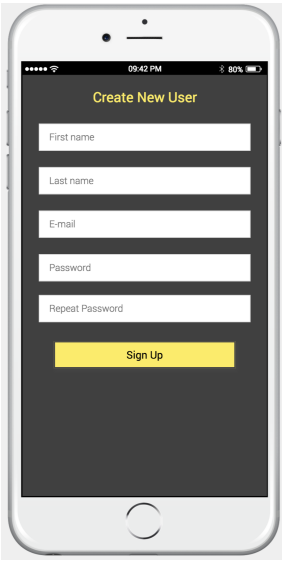
(c) Version 3

Figur 1: Log in screen

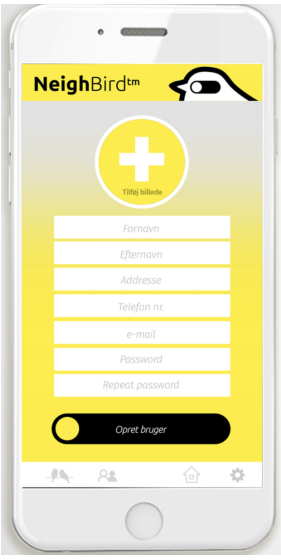
Above we see the three different versions of the log in screen. It has been very clear from the beginning how this should look and very few changes have been made. The bright yellow color, the logo and the name have been there from the start. Also, it has always been the idea that you will log in using your email address and your own personal password. The main difference is the chosen language which changed from English to Danish.



(a) Version 1



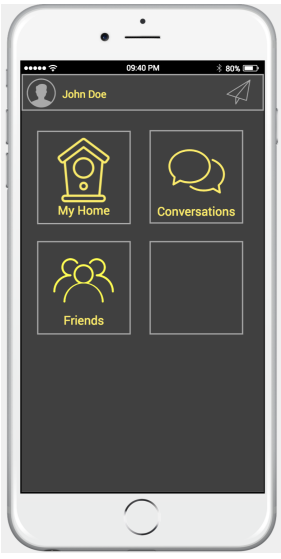
(b) Version 2



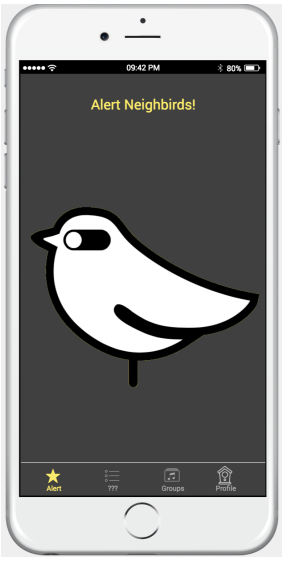
(c) Version 3

Figur 2: Sign up screen

Here we see the different versions of the sign up screen. You can access this from the log in screen by choosing "Ny bruger". The required information to sign up has not changed too much. It has been added in version 3 that it is required to fill out information about where you live seeing as it is an essential part of the whole app. Design wise a slider button was added in version 3 to implement the fun idea from the bird logo.



(a) Version 1



(b) Version 2

(c) Version 3

Figur 3: Menu and home screen

At last we see the three versions of the apps home screen - the first screen you will see when you have logged into the app - and how the menus have been setup up. At the start we went with different buttons symbolizing different menus you could access. This was quickly discarded seeing as it took too long to actually sent out an alert. Instead a tab bar menu was added in version two. This means that the first screen you will see is the place where you can send out an alert. Exactly how this would work was not completely figured out in version two. But in version three

the "how to send alerts"had been made. You will have two drop down menus; one with a predefined message, and one with groups. Then underneath we have the slider button which will sent out the alert to the specified group.

3.1.2 Final Design

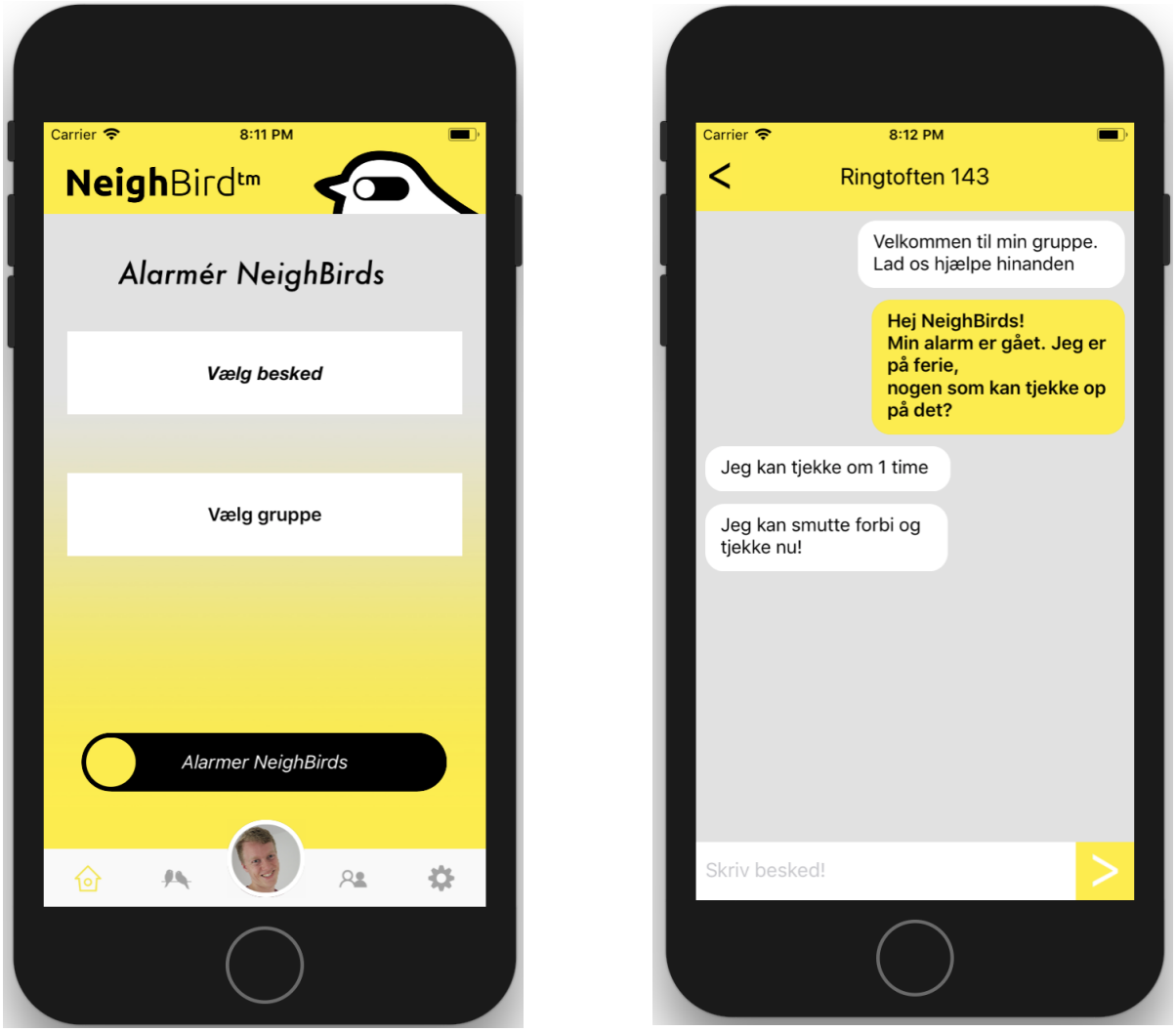
In this section we will talk about the final design of the app. It does not differ that much from the final prototype we made, so some screens will look the same. In addition we show a few more of our menus and talk about how they work.

(a) Log in screen

(b) Sign up screen

Figur 4: Log in and sign up

Above at the left we see the log in screen. From here you are able to log into the app using your email and password as we see are filled out. If you have forgotten your password you can tap the button "Glemt password?"which will take you to a new screen. Here you can enter your email address and you will be sent a link to reset your password. If you are not yet a user you can choose to tap the button "Ny bruger"which will take you to the sign up screen which we see on the right side of the figure. Here you have to fill out all the information in the white text fields. You can also add an image to your profile by tapping the round plus. When all information is filled out you simply swipe the bottom button and your profile is created. If any information is not filled out correctly an error message will be shown.



(a) Send out alerts

(b) View alerts and messages

Figur 5: Send alert and message view

Above to the left we see the first screen that will show when you log in. This is the most important one as it is here you send out alerts - the menu icon for this screen is the bird house. To send an alert you choose a predefined message from the drop down menu which will show when tapped "Vælg besked". Next you choose a group from the group drop down menu. At last you swipe the button in the bottom like you would do when creating a profile. The alert will then be sent out to everyone in the group, which is what we can see in the figure to the right. This menu's icon is the two birds. The alerts will be displayed in a chat view. The alert's background is yellow to make it stand out. Then user's of the group can answer the alert by typing a message to the group.

(a) User's profile

(b) User's list of groups

Figur 6: Profile and groups

Above figure to the left shows the user's profile. To get to the profile you tap the round image of yourself which is located in the middle of the tab bar menu. Here you are able to view your information and also edit your profile by tapping "Rediger". This will take you to a new screen where you can change e.g. your address or your profile picture.
The image to the right shows your groups. The menu icon for this is the outlines of two persons. Here you can see a list of the groups you are in or you can add a new group by tapping the yellow round plus button. This will take you to a new screen where information about the group is filled out.

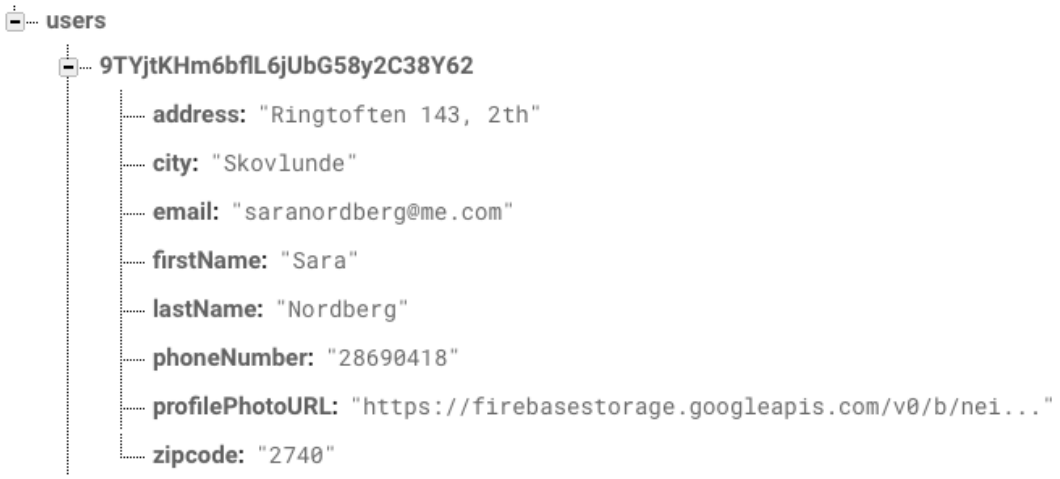
Further more we can see from the tab bar menu that one last icon exist on the far right. This is the settings page. So far it contains information about the app, and a button to log out.

3.2 Firebase

To store data we have chosen to use Firebase. Our Firebase is structured using simple objects based on their function. The main objects are; users, groups, and messages.

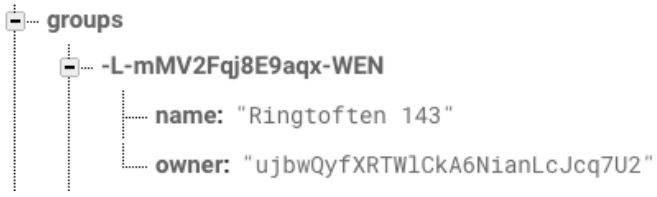
Figur 7: Firebase Structure

As seen above our Firebase is structured in a nice and clean way which allows for easy understanding and leave room for little error.



Figur 8: Firebase User

This a snippet from our Firebase database showing the values set for an user. An user object contains personal information about a person which is used to show their name when selecting members for a group. Address and phone number can be used when sending an alert as other users need to know where to go and how to contact a person if the chat is not sufficient.



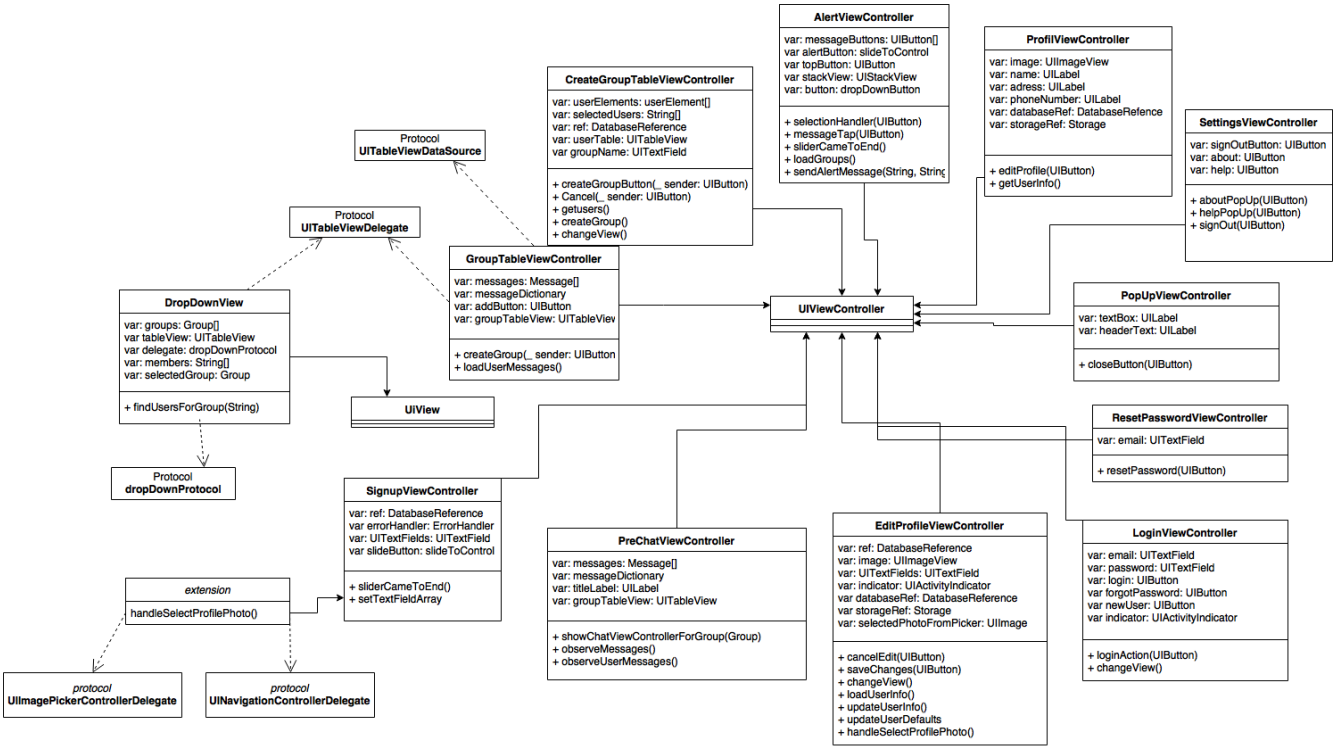
Figur 9: Firebase Group

Above is a snippet of a group object, which contains a name and an user who is the owner. The connection between the users and groups are based on two other object called user-messages and group-members. User-messages create a connection between a group, the group's members, and a newly created message. Group-members is an object which contain the members for a certain group.

Figur 10: Firebase message

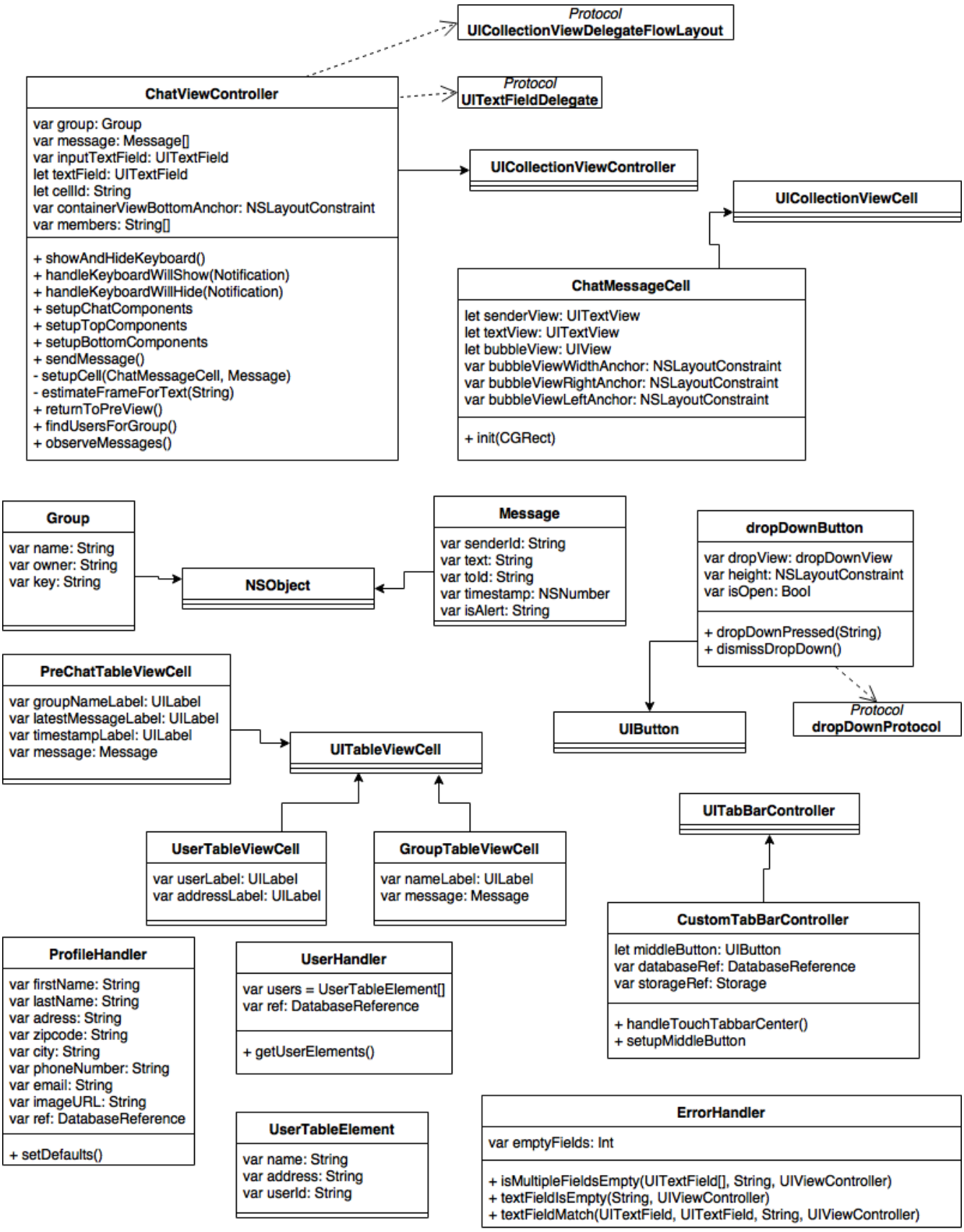
In the last snippet we see the structure for a message. Its values are simple and each have their purpose. SenderId and toId contains information about the sender and the target group. Text contains the actual message and timestamp is the exact timestamp for when the message was created. The last value is *isAlert* which determines if the message is an alert in which case it needs to be highlighted in the chat.

3.3 Class Diagram



Figur 11: Class diagram part 1

Above figure shows one part of our class diagram. This is mainly all our view controllers which handles their own part of the app's user interface. In the middle we have the class UIVIewController which all our view controllers are subclasses of. This handles the views and responds to user interactions. Some of our views uses protocols as they need additional functionality. These can be seen on the left.



Figur 12: Class diagram part 2

Figure 12 shows the second part of our class diagram. At the top we see our ChatViewController which is a subclass of UICollectionViewController due to the fact that chat is a collection of messages. It also has two protocols. In the middle we have various classes which are subclasses of respectively UITableViewcell, UIButton, and NSObject. In the bottom we see our handlers. These classes bring functionality to various viewControllers throughout the app.

4 Implementation

4.1 Frameworks

In this project we have made use of the Cocoapods framework which has provided us with simple ways to install SDKs. Cocoapods allow for easy management and oversight for additional frameworks that might have been installed onto the project. We have used it mainly to help with our Firebase integration which has been one of the main functions of this application. Firebase has allowed us to create users, store, and load data across devices.

We have also used UIKIT in a lot of classes. It constructs and manages the user interface. We have mainly used it for user input such as input from the keyboard or every time a button is being touched.
UserDefaults from the framework Foundation has been used as well. It has been used to store user information so it was accessible in multiple views.

4.2 Navigation

Figur 13: Storyboard

Above image shows our storyboard. It gives an overview of all views within the app, and gives a good idea about the navigation of the app as well. At the left we see the start view indicated by the arrow going in towards the view. This is the log in screen and from here we can see the navigation to either *forgot password* or *sign up*. Next to the log in view we see the view which sets up the tab bar menu. From this we see the navigation to our five different menus and their views. Navigation using the tab bar is implemented using the UITabBarController. Around these five menus we have different views which is mostly edit views which is connected through code to a given menu view. The only view which has not been implemented through the storyboard is the chat view which can be seen in the center bottom of the image. All objects in this view has been implemented in the code since this was the easiest way to fit our needs.
Views that is not connected with arrows in the storyboard are accessed through the code with the use of the method *present*.

4.3 Firebase

As backend we have been using Firebase because it is easy to implement and use. The user registration provided by Firebase fully fitted our needs as we wanted a simple email and password log in. Additional info was easy to create as we create user objects in Firebase by the user ID provided when first creating an account. We have created some simple values for each user which reflect the information needed for our core function to work.

Code snippet from SignUpViewController

```
Auth.auth().createUser(withEmail: email.text!, password: password.text!) { (user, error)
    if error == nil {
        print("Signup successfull")
        let user = Auth.auth().currentUser!.uid
        let email = self.email.text
        let firstName = self.firstName.text
        let lastName = self.lastName.text
        let address = self.adress.text
        let zipcode = self.zipcode.text
        let city = self.city.text
        let phoneNumber = self.phoneNumber.text
            if self.photo.image == nil {
                self.photo.image = #imageLiteral(resourceName: "Bird")
            }
        let photoName = user
        let storageRef = Storage.storage().reference().child("\(photoName).png")
            if let uploadData = UIImagePNGRepresentation(self.photo.image!) {
                storageRef.putData(uploadData, metadata: nil, completion:
                { (metadata, error) in
                    Check hvis der sker en fejl ved upload
                    if error != nil {
                        print(error!)
                        return
                    }
                    print (metadata!)
                    if let profilePhotoUurl = metadata?.downloadURL()?.absoluteString {
                        self.ref.child("users").child("\(user)").setValue(
                        ["firstName": "\(firstName!)", "lastName": "\(lastName!)",
                        "email": "\(email!)",
                        "address": "\(address!)",
                        "zipcode": "\(zipcode!)", "city": "\(city!)",
                        "phoneNumber": "\(phoneNumber!)",
                        "profilePhotoURL": profilePhotoUurl])
                    }
```

```
            })
        }
```

The screenshot above is from our code which is executed when a user has filled out our registration form and pressed "Gem". The first line is Firebase's *CreateUser* function which creates the user for our application. It takes the values from the email and password fields and creates a user. Afterwards we check that no error occurred when calling this method. If this is the case we save each field's value in a let variable. We then save the user's selected image to the Firebase Storage and saves a let variable with the download url for the image. Then all this information is used to create a new "user"object in our Firebase and the user is finally fully created with both log in and personal information.

Code snippet from ChatViewController

```
@objc func sendMessage(){
    print(inputTextField.text!)
    if inputTextField.text!.isEmpty{
        return
    } else {
        let ref = Database.database().reference().child("messages").childByAutoId()
        let sender = Auth.auth().currentUser?.uid
        let toId = group!.key!
        let timestamp = Int(NSDate().timeIntervalSince1970) as NSNumber
        let values = ["text": inputTextField.text!, "senderId": sender!,
        "toId": toId, "timestamp": timestamp, "isAlert": "N"]
        as [String : Any]
        ref.updateChildValues(values)
        let messageRef = Database.database().reference()
        for user in members {
            messageRef.child("user-messages").child(user)
            .updateChildValues([ref.key: 1])
        }
        inputTextField.text = ""
    }
}
```

This bit of code is what is executed whenever a user sends a message to a group from the ChatViewController. This is our selector which first checks that the *inputTextField* is not empty as we don not want to send empty messages. If it contains actual text we then create a reference to Firebase accessing the child node of *messages* and choose to create a new child by using *childByAutoId()*. This creates a new instance of message in Firebase and then we set the properties. We store the needed information in variables which are a senderId - that is the user's uid, a toId - which is the current group's key value, and a timestamp. The values are then stored in an array. Using the Firebase reference we then update the newly created child's values using our array. A new Firebase reference is then created to update the *user-messages* child node for each member of a group. This is used when we populate tableViews with the group for each user and display the newest message. At the end of the function we set the *inputTextField* text to an empty string so the user can type a new message.

4.4 Third Party Code

The slider button in the sign up view and the alert view are borrowed from another developer. It is owned by Christian Moler and can be found at `https://github.com/ChristianMoler/SlideTo`. He had created a slider button which fitted our needs completely and it is free to use so we integrated his code into our project. It was also very easy to modify the .xib file so the layout matched the rest of the app. The files which we borrowed from Christian are *SlideToControl.swift* and *SlideToControl.xib*. These two files have allowed us to create an interactive user experience which fits perfectly with our customers desire to have buttons matching our NeighBird logo.
The code is used in various viewControllers where they perform some kind of Firebase action. In signUpViewController when the user wants to create their profile they simply swipe the button or when a user wants to send an alert after picking a message and group they slide for it to be sent.

```
func sliderCameToEnd(){
        var message: String = ""
        if errorHandler.isMultipleFieldsEmpty(uiTextFieldArray: textFields, message: "Udf
            slideButton.setThumbViewX()

        } else {
```

Above is an example of how we implement the functionality of a sliderbutton. First we create an instance of the button in our storyboard. Then we set the view to be a SlideControlDelegate and use the function called *sliderCameToEnd*. Within this function we specify what we want to happen when the button is pulled all the way to the end.

5 Test

In this section we will go over the tests that have been performed while developing this project. The application has mainly been tested on an iPhone 7. We have also tried to test the app on an iPhone 7 Plus to make sure that it works with different screen sizes. We have chosen to test on a 4,7"and 5,5"screen as these are the most popular screen size for Apple users[1]. Only in two instances were we unable to adjust some text fields when using a 5,5"screen.

5.1 Test User

To test the app it is either possible to make a new user or use an already existing user. If the latest is the best option it is possible to use the test account. Log in details are:

- Email: test@gmail.com
- Password: testtest

5.2 Manual testing

We have done manual testing throughout the whole process to see how the app looked and reacted when being used. Every time something has been added like navigation or Firebase references we have tested that these perform in accordance to what was intended. Testing has been done using both a simulator and an iPhone to make certain that modifications made sense on a real device and not just a simulator.

5.3 Firebase testing

Several tests of the Firebase database have been performed throughout the development of the app. Every time a new Firebase function is implemented tests are needed to make sure that it behaves correctly. Luckily Firebase as well helps whenever an error accours, so testing the Firebase is very simple and easy.

[1] http://www.businessinsider.com/apple-iphone-most-popular-model-newzoo-chart-2017-7?r=US&IR=T&IR=T

6 Conclusion

In conclusion we have made a working app, NeighBird. It is possible to sign up and afterwards log in to the app with a personal user. In the app you are able to create groups and be a part of other user's groups. When you have a least one group you are also able to send out an alert, which the NeighBird group can then answer. At last you are able to view your profile and edit this if needed.
In all we feel like we have made a great app that meets the requirements we set in the beginning.

6.1 Further Improvements

Not all functions in the app has been made. We have chosen to focused to get the core functions to work, and feel as we have accomplished this. Here we will talk about a few things that could be implemented in the future.

Improvements to already implemented functions
A flaw that appears in two of our views is the keyboard overlaps the field that the user is currently editing. This is something to be improved as it is difficult to write something if you cannot see that you're writing. The issue was resolved in the chat view but the other views are still affected by this issue.

At the moment you are able to create a group in the app, but you are not able to edit it. It would be ideal to first off select a given group and afterwards edit it or add new members to it- of course only the owner of the group should be able to this.

It should also be changed how you add new users to your group. At the moment you can select a user from a list of all users of NeighBird. This is not optimal if the user database gets to big, and also it makes it easy for everyone who downloads the app to find people and where they live without even knowing them. We would like users to be found in the app by either their phone number or their email address. Then the user should get an invite to join the group, if they do not have the app yet an invitation to download the app will be sent instead.

In the chat view all members of a group can send messages, but right now it is not possible to distinguish the sender of a message. This is because no name or image is displayed with a message. It is only possible to distinguish between your own messages and others as all incoming message appear to the left of the screen, and your own messages are to the right. Name or picture needs to be implemented so the users will be able to see who is responding to an alert.

Notification
An essential part of the app is getting a notification whenever you get an alert from a NeighBird. This should be done through push notifications, and the notification should give a quick overview of the alert. If a user then slides the notification to open it, the app should open straight into the group chat with the given alert. Furthermore quick answers could be implemented to the notification with different pre-made messages to answer the alert, so you would not even have to open up the app.

Extra info for a user
At the moment the user's info consists of their address and phone number (apart from the required info to sign- and log in to the app). We would like to also add different specifications about the place people live. This could be if they live in an apartment or a house, and how many entrances the home has. It could also consist of the type of security system the user has. All this info will be helpful to your NeighBird groups if they have to check your house, and see if a burglar has entered somewhere.

Open or closed alerts
At the moment all alerts sent to a group chat will stay there together with all other messages sent. In the future it would be ideal to set a status when the alert is sent. The start status could e.g. be "open", and when the owner of the alert finds that the help they sought has been met, they can stop this given case and the status would be "closed". By doing this we would be able to delete an alert chat when it is no longer needed. This way we will avoid having excess data laying around in the app that will never be used again.

Language
To start off we have chosen to make the app in Danish only, seeing as this would be the main target users. In the long run the app should be able to change to different language so more users would be able to enjoy the app and its features.