ITMINDS

YOUNG MINDS - EXCELLENT SOLUTIONS

OS2Indberetning - mobildokumentation

15/1-16

IT MINDS
YOUNG MINDS - EXCELLENT SOLUTIONS

Versionshistorik

Version	Dato	Beskrivelse	Ansvarlig
1.2.0	08-04-16	Android added, iOS changes and minor additions to multiple sections.	MFJ & JOC
1.1.1	15-01-16	General rules section added	JOC
1.1.0	22-12-15	Windows Phone section added	JOC
1.0.2	21-12-15	DriveReport diagram updated	JOC
1.0.1	21-7-15	Web and iOS	JHA

IT MINDS
YOUNG MINDS - EXCELLENT SOLUTIONS

Indholdsfortegnelse

IT MINDS
YOUNG MINDS - EXCELLENT SOLUTIONS

Introduction

This document describes the system surrounding the mobile platform of the OS2 project. The document is divided into four parts. The first part describes the web application and the web services that are used to relay the information between the smartphone and the master database. The second part describes the general architecture of the Android application, and the third and fourth parts describe the iOS and WP8 applications, respectively.

Web Services

The master server, is the server the OS2 web client is using to function. This server contains sensitive data, and to avoid exposing this, the mobile app does not communicate directly with the master server. Instead a DMZ server is setup, which allows communication through port 8080 (HTTPS). Web Services are used to send data between the mobile app and the DMZ server, and the DMZ server is synchronized with the master server through a nightly synchronization.

Login and Authorization

To login into the Mobile Applications users has to create a password on their providers OS2Indberetning page, where they also find their username. This generates a unique GuId that is sent to the device upon login and which is then used when sending DriveReports and syncing user data on the devices.

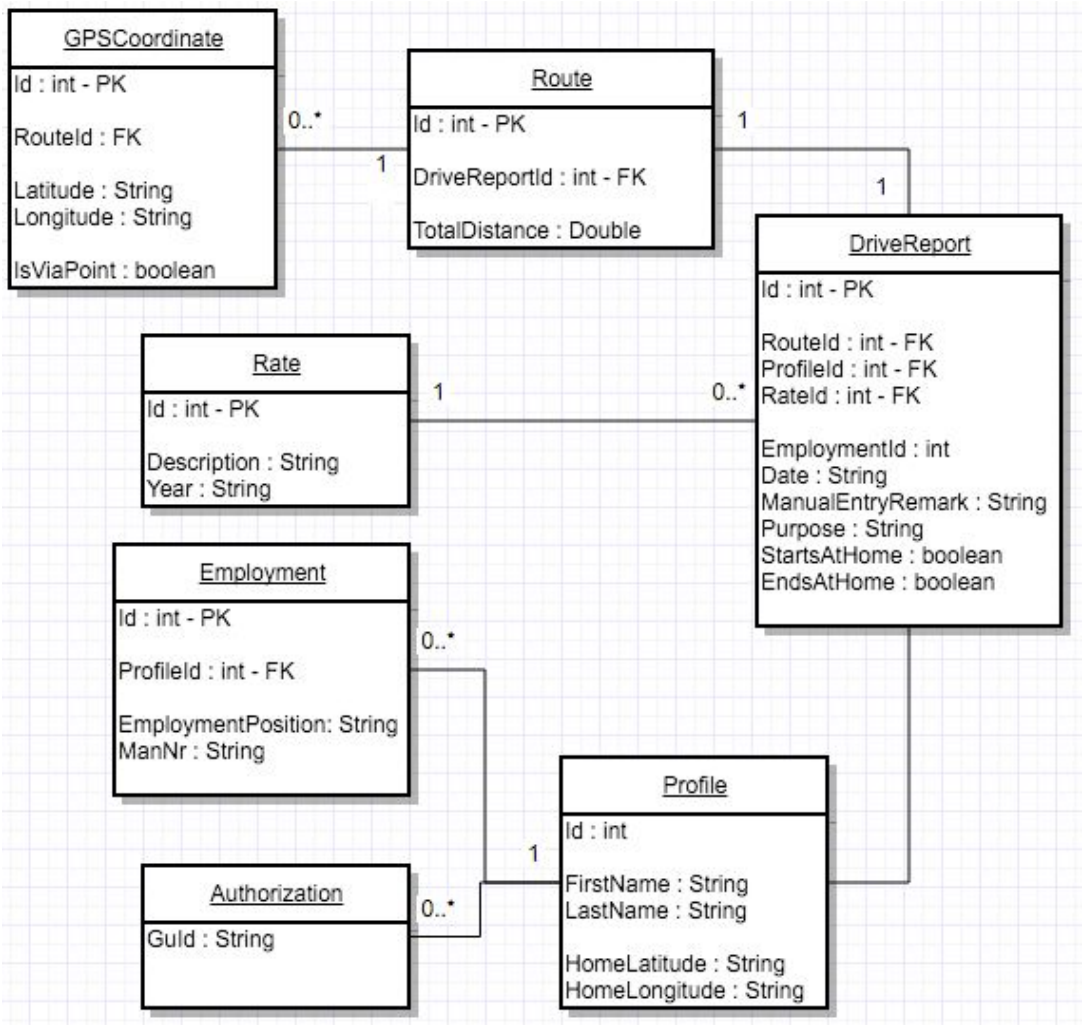
Encryption

Some of the data, which has to reside on the DMZ server, is sensitive, and has to be encrypted. This is done with a helper class named Encryptor. A encryption key is set inside encryptor.cs. This key should be unique for the installation. (and should probably be moved to a config file?).

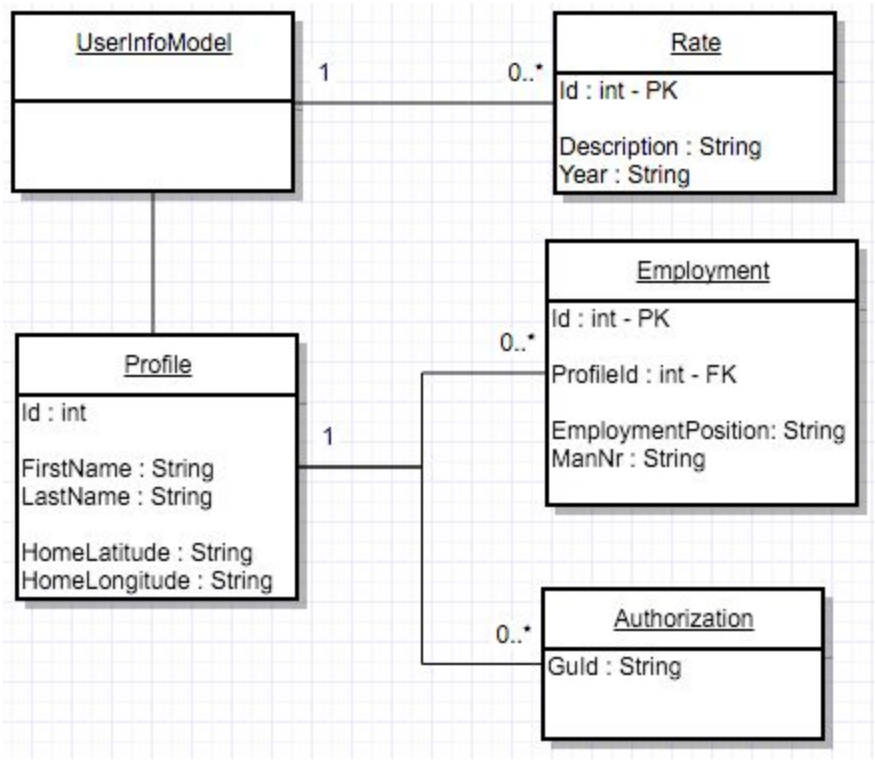
The encryption is made based on the code found here http://stackoverflow.com/questions/10168240/encrypting-decrypting-a-string-in-c-sharp, which implements AES encryption using the RijndaelManaged cryptography class.

DMZ Data Model

The data model of the DMZ server is a condensed version of the one from the main server. The DMZ data model is only supposed to hold the necessary information to create a drive report. An overview of the structure can be seen below (Focus on info that is used on Mobile Applications).



The UserInfoModel object contains information about the current user, and the possible rates. The representation of the object can be seen below.



DMZ Web Services

On the DMZ server a number of web services are available. These web services are used when communicating with the mobile application. A short description of each endpoint is provided below.

Name	/auth
Type	POST
Body params	username : String password : String
Return	UserInfoModel
Purpose	Used when the user is logging into the application. The user enters their credentials and these are sent as JSON to the server, which, if credentials are valid, returns a UserInfoModel that is stored on the device.

Name	/userinfo
Type	POST
Body params	GuId : String – from the UserInfoModels Authorization object received on login.
Return	UserInfoModel
Purpose	Used to sync the users info with the DMZ, to make sure logged in user is valid.

Name	/report
Type	POST
Body params	driveReport : Drivereport Authorization : Authorization
Return	HTTP Status code
Purpose	Used to submit a DriveReport as JSON to the DMZ. (Both new and reports saved on the device)

Name	AppInfo
Type	Get
Body Params	-
Return	Json array of AppInfo (iOS)/Provider (Android) Json Objects (see Below)
Purpose	Used when the app is first launched to list the possible servers to connect to. This will determine the URL of the server that is going to be used, and the colors of the app.

The AppInfo JSON Objects that are returned are stored as JSON data in a static file, which permanently resides on Favrskovs web server (Currently: https://ework.favrskov.dk/FavrskovMobilityAPI/api/AppInfo) The objects are defined as follows:

```
{
  "Name": "Favrskov",
  "APIUrl": "https://ework.favrskov.dk/FavrskovMobilityAPI/api",
  "ImgUrl": "http://www.denstoredanske.dk/@api/deki/files/8935/=39997926.jpg",
  "TextColor": "#FFFFFF",
  "PrimaryColor": "#00665F",
  "SecondaryColor": "#DB813C"
},
```

This is the object defining the API endpoint to use, the colors of the app and the URL for the municipality's image.

Integration with the Master server

Every night a synchronization of the DMZ and master database is scheduled.
The general structure of the synchronization is as follows:

- Transfer all drive-reports from DMZ to Master
- Sync all person data from Master to DMZ
    - Inactive people are set to Inactive in DMZ and won't be able to login
- Sync all rates from Master to DMZ

General rules

Posting rules

Before submitting drivereports to the backend, there are some rules to check for.

- Before posting your drivereport, make sure that GpsCoordinates are formatted with a dot and not a comma.
- There should always be either 0 or atleast 2 GpsCoordinates in your drivereport before posting it. 1 is Invalid
- The user has the option to edit the kilometer amount calculated by the GPS tracking. If the user chooses to do this, the coordinates in the route are discarded and no map of the route can be generated.

Gps rules

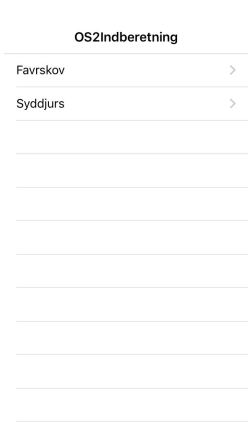
When tracking routes with the GPS, some rules have been specified that all platforms follow.

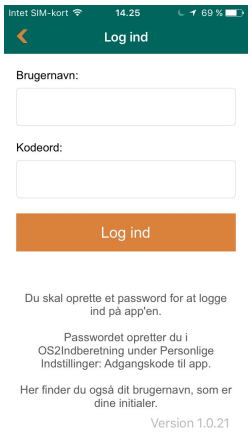
- If pause is activated, the user cannot resume, if he/she has moved more than 200 meters away from that spot where pause was enabled.
- When new measurements are received from the GPS, they are discarded if they are below the current accuracy.
- If the GPS accuracy is above 100 meters, all measurements are discarded.
- When pause is pressed, the last known coordinate is considered a IsViaPoint and is therefore set to 'true'
- If there has been no GPS signal for 30 concurrent seconds, the user is notified about possible inaccuracies in the route when he is done tracking.

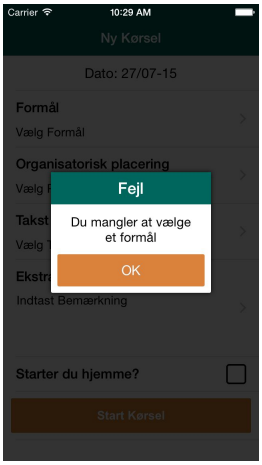
Mobile App

User Interface

This section contains screenshots from the Favrskov iOS version of the app. (There are minor differences on Android, but the general user interface and interaction is the same and only differs where each platform demands it).







Location

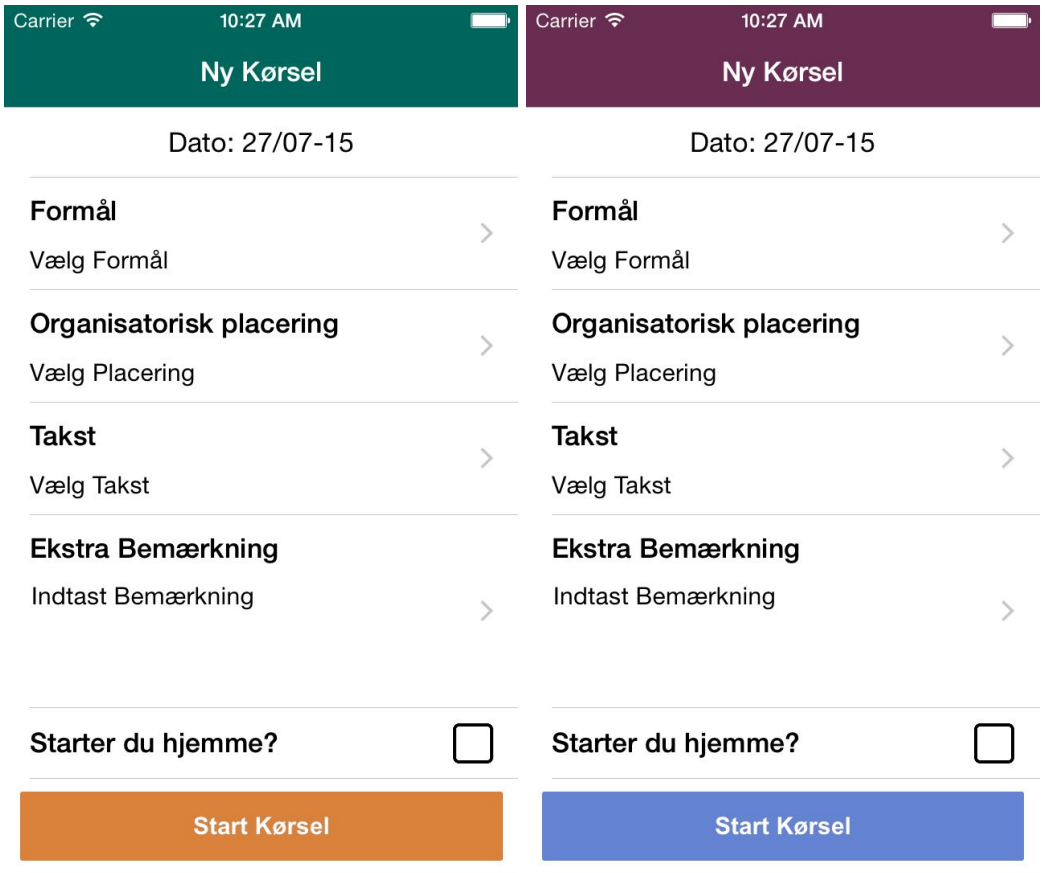
Both apps keeps track of the GPS stability and accuracy. If the accuracy is larger than the apparent distance between the last location and the current one received, we discard the point.
Furthermore, if the distance between 2 locations is more than 200m, the user is warned after the drive, that the distance might not be accurate and they should check it.

Mobile App – iOS

The iOS mobile application was developed for iOS7+ using XCode 6.4+.
The application uses storyboards for general layout, and xib files for reusable UI elements.

Setting the AppInfo JSON object described in the DMZ Web Services section above customizes the color scheme of the app. The AppInfo object allows customization of the logo used during synchronization, as well as customization of the primary, secondary and text color of the app. This information is saved when a user selects a provider before the login screen and is cleared when the user logs out.

An example of the look of the app with different colors can be seen below.



IT MINDS
YOUNG MINDS - EXCELLENT SOLUTIONS

Architecture

The app uses typical MVC. Each view is controlled by a viewcontroller and backed by model. The model used in the app resembles the DMZ data model.

CoreData is used to persist data. Because CoreData objects can be inconvenient to work with, a two layer persistency model is created. At the lowest model layer, regular CoreData managed objects are used to persist and retrieve data. At a higher level, PONSOs (Plain old NSObject) are used to work with the data.

A CoreDataManager, a HTTPS Client, a GPS Manager and a UserInfo object are all created as singletons, as they all require a single instance.
The CoreDataManager is a wrapper around CoreData, The HTTPS client is a wrapper around AFNetworking, the GPS Manager is a wrapper around CoreLocation and the UserInfo object is a convenience class for user specific data, which is stored in NSUserDefaults.

CoreLocation

The tracking is done using CoreLocation, which is wrapped by the class GPSManager.

The activityType is set to kCLLocationAccuracyBestForNavigation, and the distance filter is set to 10 meters.

Pods

Cocoapods was used for dependency management. The external dependency used in this project, AFNetworking, is used to for interacting with the web services.
The content of the podfile is shown below:

```
xcodeproj 'OS2Indberetning.xcodeproj'
platform :ios, '7.0'

pod 'AFNetworking', '~> 2.0'
pod 'XcodeCoverage', '~>1.0'

target 'eIndberetningTests', :exclusive => true do
  pod 'KIF', '~> 3.0', :configurations => ['Debug']
end
```

GPS Mocking

A set of files with a premade coordinates are located in the project All these files has the extension *.gpx. These can be used to let Xcode mock location on a device.

Mobile App – Android

The Android application was developed for Android 4.0+ using Android Studio and Gradle for dependencies.

The Android version, like the iOS version, also has an object used for customization of the color scheme and other municipality specific settings. This is the Provider class. Usage of the info stored in the Provider class is mainly handled by the ColorHandling class.

For an example regarding the color difference, see the ‘Mobile App - iOS’ section.

Architecture

The app is build like most Android apps, with Activites handling the views, which are backed by the model and helper classes. As in the iOS version the model used in the Android app resembles the DMZ data model.

SharedPreferences is used for local storage and settings. This is all done via POJO -> JSON for saving and JSON -> POJO for reading. The classes representing the model has the methods for the reading conversion. The responsibility for this is the MainSettings class.

Location

The location tracking is done using Google Play Services - Location. The LocationMgr handles the interaction with the system service and runs in the background. This is then wrapped inside the GpsMonitor, which handles relaying of data to the UI.

Settings for the GPS is PRIORITY_HIGH_ACCURACY.

IT MINDS
YOUNG MINDS - EXCELLENT SOLUTIONS

Gradle

Dependency management is handled via Gradle. For communication Volley and OkHttp was used. For location Google Play Services. For easier date handling Joda was used.

Snippet of the dependencies are shown below:

```
compile 'com.android.support:appcompat-v7:23.2.1'
compile 'com.mcxiaoke.volley:library:1.0.19'
compile 'com.squareup.okhttp:okhttp:2.6.0'
compile 'com.google.android.gms:play-services-location:8.4.0'
compile 'net.danlew:android.joda:2.8.1'
compile 'com.android.support:design:23.2.1'
```

GPS Mock

Mocking locations on Android was done with external apps (E.g ‘Mock Location’), resulting in no local files for location mocking.

Mobile Application - WP8

The Windows Phone application is developed using Xamarin forms. We originally targeted the WP8.1 platform, but found out that 8.1 does not allow background tracking. The project was downgraded to 8.0 to allow this. Both platforms should be equally compatible with windows phone 10.
On Git the 8.1 project is kept in a “outdated” branch. This branch is kept for reference. Do not attempt to merge this into the main branch.

Because 99% of the code is kept in a portable class library, native implementations is handled by Xamarin and other frameworks. Xlabs have made alot of these implementations, like a geolocator class and a secure storage class that we use, instead of implementing our own native solutions.

Architecture

The WP8 application makes use of the MVVM pattern. This works by hooking up a page and a viewmodel using the xlabs.ViewFactory. When this is done, new views can be pushed onto the stack by using Navigation.PushAsync<theViewModel>(). Note that pushing a type is only doable from the viewmodels because they inherit from a xlabs mvvm class. The normal syntax would be Navigation.PushAsync(new page), and that page would probably new a viewmodel and set it as its bindingcontext to that viewmodel. When using the ViewFactory the bindingcontext gets set automatically.

The Page and the ViewModel has no direct knowledge of each other. They send messages through the message system supplied by Xamarin. If there are cases where sending messages isnt good enough, access to the viewmodel can be obtained by using the (bindingcontext as someViewModel).SomeMethod for accessing methods in the viewmode, from a page.
When viewmodels receive messages they are supplied with the pages context, and can thereby access public methods and properties if need be.
Model binding works using a SetBinding method.

User Interface

The UI is created in code behind. It is possible to make it in XAML but that path was not chosen. The application is made to look more or less like the iOS version.

Buttons

All buttons in the application are a little special. They are not ordinary buttons. The reason for this is because the standard WP buttons changed the background of the button to blue when it was pressed. This looked really bad in some places where the background of the button should match the background of a header. To get around this, all buttons are basically a stackpanel that wraps other elements, and catches a tap gesture. When the tap gesture is captured, it calls a callback function that you specify. It is also animated to do a “pressed down” animation.
The downside to this approach is that these custom “buttons” cant be binded to a command property. They need to receive a callback to do a specific thing. In our case these callbacks are usually functions that just send a message to the viewmodel via the messagingcenter.

The positive side is that these buttons are very easily customizable

GPS Tracking

The GPS Tracking is done using xlabs geolocator implementation. When the user chooses to start or resume the location tracking, there is a 5 second “countdown” before the tracking starts happening. This is due to the inaccuracy in the first couple of seconds. Without the countdown, the GPS could easily have tracked 30-60 meters without any movement.

Loss of gps signal

If there is a loss in the gps signal, a timer is started. this timer will run as long as there is no signal. If there have been no signal for more than 30 straight seconds, the measurement is considered inaccurate. This means that the user will receive a popup message telling him to be aware of the lost signal when he submits hit drive report. If the signal returns after 15 seconds, the timer is stopped and reset. Meaning the the user can lose signal for multiple times without receiving a popup notification about it, as long as the timeouts are shorter than 30 seconds.

When there is no signal the user can see it on the screen where the gps accuracy usually is.

Data storage

Secure Storage

All sensitive data is stored on the device using xlabs SecureStorage. The Secure storage takes a byte array, so the data is serialized or deserialized before or after usage of the secure storage.

File data

If uploading of drive reports fail, the user gets the option to same those reports for upload at a later time. Those reports are stored in the file system, as a text file.

Static data

A lot of temporary data is also stored in a static class called Definitions. Things like screen height and width are stored there. Color schemes and storage keys. The DriveReport is also stored there. This is done so that its reachable by all views without the need to inject.

Testing

Xamarin offers automated UI test for both iOS and Android, but unfortunately not for WP at the moment. Microsoft does have an automated UI test to XAML based Windows phone 8.1+ applications. Therefore, there is very limited testing done in the original 8.1 project. The 8.0 project currently contains no testing project.