Project Info

Team “Sky”

Virtanen Samuli
samuli.virtanen@metropolia.fi
Student ID: 1706048

TABLE OF CONTENTS

1. PROJECT DESCRIPTION

Project idea was to create a Flight Recording app that uses mobile phones internal sensors to log the flight data (location, altitude, speed, start and end time, takeoff and landing time and also landings amount. This application has a real use-case and need – so it was nice to be able to do this kind of project.

2. INSTALLATION & USING

2.1. INSTALLING

1. `git clone https://github.com/samuliv/sky-fdg-android/`
2. Run on Android Studio

2.2. TEST LOGIN CREDENTIALS

Username:	`testi`
Password:	`testi`

2.3. APK-FILE (Android Application Installer)

https://www.tucloud.fi/metropolia/sky.apk (built 2019-10-11)

3. FIRST USER INTERVIEWS / TESTS

I discussed with a few pilots what kind of the application should be and created the first sketches with Figma. Got some ideas, feedback and important info for the project. More in the PDF files (see the links -section).

4. SANDBOXING THE COMPONENTS/FUNCTIONALITIES

I started the coding process doing elements of the project code in separate sandboxes (Flight Recorder, GPS logger, JSON-class, etc…). So I created and tested basic core functionalities separately and after I got core components designed in a beta state I combined those into the final first project file. Also the first flight data logging (for the algorithm design) was made with a separate, crappy-looking sandbox project FlightDataGather.

5. BACKEND DESIGNING

I created the Backend API basic structure ("core") for the "**SkyAPI**". I implemented three ways of communicating with the backend;

1. **PUBLIC REQUEST** – that does not need any authentication
2. **SESSION REQUEST** – that requires **user** logged in and token + session ID
3. **API KEY REQUEST** – that requires the API-key for the response

API-key based requests are (at least not yet) used by this project. Public requests are for example "test" and "login" requests. Session requests are widely used by the final Android App (almost all the requests excluding "login").

All the data that App uses is coming from the Backend.
Backend API readme in Links-section.

6. USER TESTS AND TEST FLIGHT (WITH THE APP)

I did two separate set of User Tests. First User Test took a place in when I was about to start the actual design of the App. Second test, when the first working prototype with Android Studio was made.

Main points of User Tests:

- App looks like nice and it is easy to use
- No day-mode-view required
- Contrast easy-changing would be nice → **TODO**
- Auto-detect Aircraft is nice feature
- IOS -version is needed also (one of the pilots said he'll change to Android...)

Main points of Test Flights:

- First test flight was for data gathering for algorithm designing
  - Worked well, nothing extra to say about that, app was stable
- Second test flight was for testing the app Prototype
  - App worked well
  - Algorithm didn't catch all the events because of inaccurate altitude info
  - Algorithm needs thinking and further development
  - App was Stable
  - You can not trust on GPS altitude, how about Barometric alt sensor?
    - TODO: development → Baro-sensor testing/test flights...

UI/UX tests separate pdf files in Links -section.

7. PROJECT STUCTURE INFORMATION

7.1. IN SHORT

- Everything “viewed section” is a Fragment
- Sometimes you see multiple fragments in a screen
- Sometimes fragments can contain fragments inside…
- Every string is read from **strings.xml**
- Tried to use preferred ways to name folders/variables/layouts/drawables…
- No linter warnings – clean code & structure – private when needed

7.2. FOLDER STRUCTURE

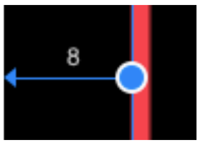
activities		Activities.
fragments		Fragments.
services		Services (runned as Service).
ui.main		UI main stuff (basically adapters)
	adapters	Adapters for RecyclerViews
utils		Utilities / Classes / Objects
	algorithms	App algorithms.
	databases	SQLite local database objects and implementations.
	dataclasses	Common Data Classes
	enums	Common Enums
	interfaces	Common Interfaces
	jsonrequest	JSONRequest -API class separated here totally
	properties	App Constant Values and Properties
	simulation	SimulationClass separated here (to be deleted when published)

7.3. MODULE EXPLANATIONS

FlyAlgorithm	Extends FlightRecorder class and adds automation for listening flights events (takeoff, landing) and performs operation logging of those events automatically.
JSONRequest	Custom-made JSON-request class for Backend API communication and JSON-response data parsing.
SimuFly	Flight Simulation class for testing (will be deleted in the released app version).

Backend	Implements Backend-calls and uses JSON.request
BluetoothBeaconScanner	Aircraft BLE Beacon Scanner for determining automatically which plane is being used.
FlightRecorder	Flight Recorder basic class – records a flight and provides main functionality for that
GPSMath	Mathematical functions around GPS-stuff. For eg. calculate distance between two GPS location points, isNearWaypoint and flown distance calculation
ISOTime	Provides time formats (in ISO format – and other formatting for time)
PreferenceHelper	Provides SharedPreference using
Session	Handles the session by using SharedPreferences
Settings	Provides a middlemen for requesting setting values from PreferenceHelper (some requests can be made often so this keeps some most used values in memory, without using SharedPreferences in every check)
UI	UI functions (button enabling/disabling in easy way, toast..)
WaypointLocations	Keeps waypoint info in a memory because these data is checked many times during the flight recording process – so database calls have not to be made in every second

8. MATERIAL DESIGN / ACCESSIBILITY / UI



Tried to follow the basic Material Design guidelines and make this app to look like and act in a way that typical Android Application does. Basically my UI is a bit experimental looking, but it still follows the basic MD rules and Android application logic.

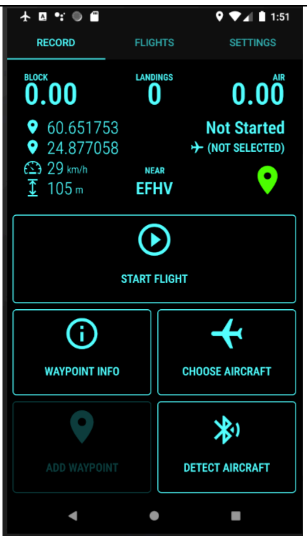
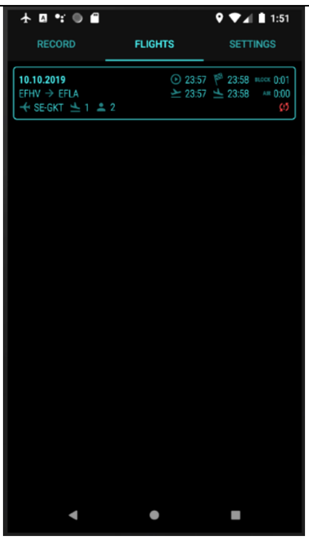
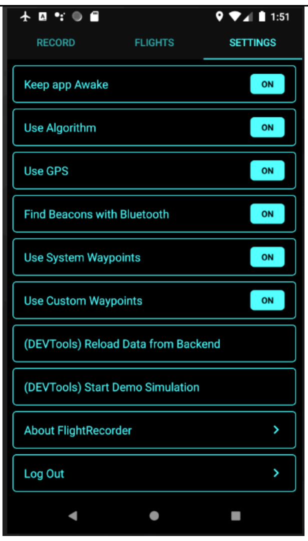
I understand the accessibility needs and tested my app with the Accessibility Scanner also. I check all the warnings the scanner gave me and checked out does it make any accessibility issue when using this product (implemented also the descriptions for the images – just for the learning and test purposes). Still, in this apps context the users are pilots and have Grade A-vision and health condition, so basically those do not need any Accessibility Features (like screen reader or something like that). Accessibility in this context is about this have to be easy to use during the flight, for.eg.;

- big/easy buttons to press – if buttons have to be pressed during the flight
- sound notifications are out of the question – too much noise during the flight (have to use ear protection headphones – could be usable – but hard to implement anything connectable to those – headphones are analog and connected to flight radios/equipment)

- visual notifications are also mostly out of the question – a pilot cannot just stare into a phone screen during the takeoff/landing situations ☺ they have to focus on flying in those situations – so best way is to make the algorithm work ~100%....
- Black theme "negative", because the app have to be usable at night. I also asked the pilots, is it necessary to make different UI for daytime – and everyone said that this theme is ok for both of the situations.

9. USER INTERFACE EXPLAINED

APP MAIN VIEWS (TABS):

		
Record Tab This section is used during the flight recording process.	(Recorded) Flights Tab This section is used for viewing recorded flights / edit information of those	Settings Tab This section is used to change the app settings.

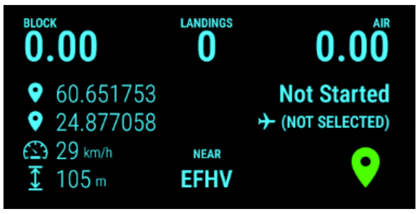
RECORD FLIGHTS SETTINGS

10.10.2019 23:57 23:58 BLOCK 0:01
EFHV → EFLA 23:57 23:58 AIR 0:00
SE-GKT 1 2

RECORD FLIGHTS SETTINGS

Keep app Awake ON
Use Algorithm ON
Use GPS ON
Find Beacons with Bluetooth ON
Use System Waypoints ON
Use Custom Waypoints ON
(DEVTools) Reload Data from Backend
(DEVTools) Start Demo Simulation
About FlightRecorder ›
Log Out ›

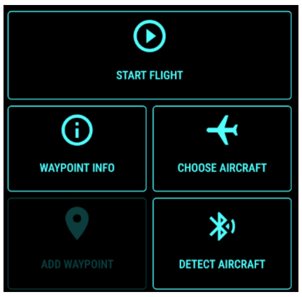
RECORDER DISPLAY



- Time formats are explained in the whole app like "MM.SS", "HH:MM:SS" or "HH:MM" (dot separates minutes . seconds, hours : minutes)
- Block = total flight time, Air = time on air
- Location coordinate / altitude / speed formats can be changed by tapping
- "Near EFHV" displays that you are near to (ICAO-)waypoint called "EFHV"
- "Not Started" displays flight recording status
- "Plane: (not selected)" displays plane in use

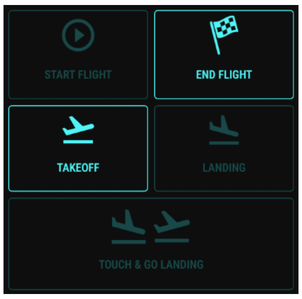
- Location Icon: (GREEN=Located, RED=(blinking=not located, stable=gps off))
- Landings: amount of landings

BUTTONS (BEFORE RECORDING STARTS)



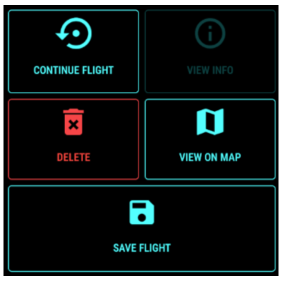
- START FLIGHT = start recording
- WAYPOINT INFO = get info about the waypoint you are near to
- CHOOSE AIRCRAFT = select plane you're flying with
- ADD WAYPOINT = create new custom waypoint
- DETECT AIRCRAFT = use Bluetooth for finding a beacon inside a plane to determine used plane automatically

BUTTONS DURING THE FLIGHT



NOTE: If the algorithm works 100% you should only need to press the "END FLIGHT" - button!

BUTTONS AFTER THE FLIGHT RECORDING (after pressing END FLIGHT)



CONTINUE FLIGHT = go back, if you pressed END FLIGHT accidentally

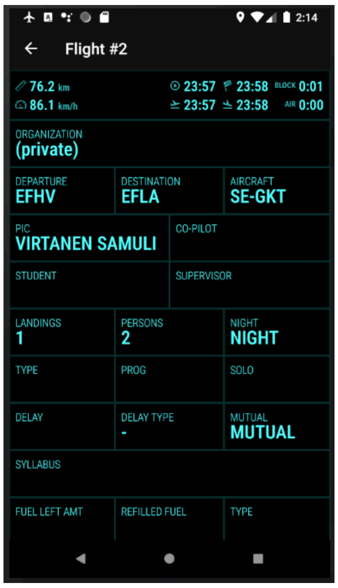
VIEW INFO = view info about recorded flight (TODO)

DELETE = delete recorded flight

VIEW ON MAP = View Flight on a Map View

SAVE FLIGHT = save flight into a Local SQLite database (store it permanently)

FLIGHT EDIT VIEW



2:14

Flight #2

76.2 km
86.1 km/h

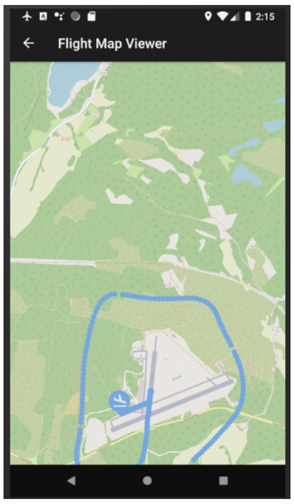
23:57 23:58 BLOCK 0:01
23:57 23:58 AIR 0:00

ORGANIZATION
(private)

DEPARTURE EFHV	DESTINATION EFLA	AIRCRAFT SE-GKT
PIC VIRTANEN SAMULI	CO-PILOT	
STUDENT	SUPERVISOR	
LANDINGS 1	PERSONS 2	NIGHT NIGHT
TYPE	PROG	SOLO
DELAY	DELAY TYPE -	MUTUAL MUTUAL
SYLLABUS		
FUEL LEFT AMT	REFILLED FUEL	TYPE

Enables user to extend / correct recorded flight info before storing/sending it into the backend server.

FLIGHT MAP (RECORDED ROUTE) VIEW



Shows the start and end location of the flight on a map.

Shows the flight route on a map. Image above showing the landing part of the recorded flight.

TODO: Implement dark mode for the map.

10. NEXT STEPS / TODOS

- Fix the known bugs/issues
- Do the TODO -things
- Test application more for error situations
- Algorithm further designing
- Beta release for real testing
- More coding/fixing/User Tests/testing
- Public Release
- Coding IOS-version of the app for Apple devices ☺

11. TECHNICAL SPECIFICATIONS

11.1. API TARGET (MIN SDK)

- API TARGET IS >= 16 – so this will work in almost every device ☺
    - This was also a reason why modern MD/UX/UI features weren't available ☹
- **Note:** BLE (beacon scanning) is only supported if the API Level is >=21

11.2. EXTERNAL (ANDROID) LIBRARIES USED

- ROOM 2.2.0-rc01
- ANKO 0.10.8
- OSMdroid 6.1.0
- MapsForge 0.12.0 (not in use in the app currently – but in testing)

11.3. IMPLEMENTED/USED FEATURES

- Internal GPS sensor using, Baro-sensor in planning/testing for the app…
- Material Design Guidelines
- Broadcast Receiver / sending broadcasts used in the whole app design to pass messages around
    - TODO: Implementation of listening Network Statuses – could be helpful for not to do unnecessary requests if the network is not available or Flight Mode is activated…
- Service for the Flight Recording to work in the background also
- Data storing internally with ROOM/SQLite
- Data storing internally with SharedPreferences
- Clean Code / Package Structure – at least tried to…
- Bluetooth BLE Beacon finder – to determine a plane that is currently used

- OSMdroid map view for viewin the recorded flights
- Backend API communicating (HTTPS/Custom Made Backend Service)
- ViewModel/LiveData used

11.4. BACKEND

- Custom made Backend API for this project
- Provides all the information for the application (Waypoints, Users, Aircrafts, Flight Logging Item Special Types, etc.)
- PHP 7.2 + MariaDB + JSON
- Connected to the Production Server
- More information in Links…

12. FINAL CONCLUSIONS

This Project / course was fun to do. It taught me a lot about Android App designing (how to do the things in Android, Usability/Accessibility, MD, colors…). It was also nice to be able to do something, that could be in real use some day – and it fit also the school requirements.

The time was still short, and the project was coded a bit in a hurry, however, I was pleased with the result.

I also came up with some new ideas during the project, for example “a beacon tracked plane near can be used to send Backend the data where the plane currently is physically” – a fun feature in the Aviation System to see where the planes real exists currently. ETC…

13. LINKS

Project Info GitHub:

https://github.com/samuliv/sky-fdg-project

Project Management:

https://github.com/samuliv/sky-fdg-project/projects/1

First Sketches (FIGMA):

https://www.figma.com/file/YJUHoxzLCpt6Qs1bNiAmTUNn/Untitled?node-id=0%3A1

First User Tests (PDF):

https://github.com/samuliv/sky-fdg-project/blob/master/FLIGHT_RECORDER_UIUX_1.pdf

Second User Tests (PDF):

https://github.com/samuliv/sky-fdg-project/blob/master/FLIGHT_RECORDER_SECOND_UX.pdf

Backend API Documentation:

https://github.com/samuliv/sky-fdg-project/blob/master/Backend-Api-Calls.md

Backend API-address:

https://www.aviatron.fi/_skyAPI.php

Demonstration Video:

https://github.com/samuliv/sky-fdg-project/blob/master/flight_simu.mov

Presentation PDF:

https://www.tucloud.fi/metropolia/readme.pdf

APK installer (proto):

https://www.tucloud.fi/metropolia/sky.apk