BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

ELECTRONIC FLIGHT BAG

ELECTRONIC FLIGHT BAG

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. GABRIEL LEHOCKÝ**
AUTOR PRÁCE

SUPERVISOR **doc. Ing. PETER CHUDÝ, Ph.D, MBA**
VEDOUCÍ PRÁCE

BRNO 2017

Master's Thesis Specification/18375/2016/xlehoc01

Brno University of Technology - Faculty of Information Technology

Department of Computer Graphics and Multimedia Academic year 2016/2017

Master's Thesis Specification

For: **Lehocký Gabriel, Bc.**
Branch of study: Computer Graphics and Multimedia
Title: **Electronic Flight Bag**
Category: Computer Graphics

Instructions for project work:

1. Perform a study on the state-of-the-art Electronic Flight Bags.
2. Research key features of an Electronic Flight Bag.
3. Design and implement an Electronic Flight Bag for Android OS.
4. Perform testing and evaluation of the developed Electronic Flight Bag.
5. Suggest future research directions.

Basic references:

- Specified by the supervisor

Requirements for the semestral defense:
Points 1, 2 and partially point 3.

Detailed formal specifications can be found at http://www.fit.vutbr.cz/info/szz/

The Master's Thesis must define its purpose, describe a current state of the art, introduce the theoretical and technical background relevant to the problems solved, and specify what parts have been used from earlier projects or have been taken over from other sources.

Each student will hand-in printed as well as electronic versions of the technical report, an electronic version of the complete program documentation, program source files, and a functional hardware prototype sample if desired. The information in electronic form will be stored on a standard non-rewritable medium (CD-R, DVD-R, etc.) in formats common at the FIT. In order to allow regular handling, the medium will be securely attached to the printed report.

Supervisor: **Chudý Peter, Ing., Ph.D. MBA**, DCGM FIT BUT
Beginning of work: November 1, 2016
Date of delivery: May 24, 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

Jan Černocký
Associate Professor and Head of Department

Abstract

This thesis describes the standards, types and use of the *Electronic Flight Bag* systems in aviation. It further discusses the design and implementation of an application based on the European regulations for *General Aviation* pilots for tablet devices with an *Android* operating system. The main features of the designed application are the flight manual, the flight publications and document browser and reader, user centered flight navigation, weight and balance calculation, and other interactive calculations.

Abstrakt

Tato diplomová práce popisuje standardy, typy a použití leteckého systému *Electronic Flight Bag*. Dále se zabývá návrhem a implementací aplikace pro tablety s operačním systémem *Android*. Tato aplikace vychází z evropských směrnic pro piloty všeobecného letectví. Mezi hlavní vlastnosti vytvořené aplikace patří letový manuál, prohlížeč letových publikací a dokumentů, letová navigace zaměřena na uživatele, kalkulace hmotnosti a těžiště, a další interaktivní výpočty.

Keywords

EFB, Electronic Flight Bag, Android, Java, aviation

Klíčová slova

EFB, Electronic Flight Bag, Android, Java, letectví

Reference

LEHOCKÝ, Gabriel. *Electronic Flight Bag*. Brno, 2017. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor doc. Ing. Peter Chudý, Ph.D, MBA

Electronic Flight Bag

Declaration

Hereby I declare that this thesis was prepared as an original author's work under the supervision of doc. Ing. Peter Chudý, Ph.D. MBA. All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

.....
Gabriel Lehocký
May 17, 2017

Contents

List of Figures

List of Tables

List of Listings

Abbreviations

AIP	Aeronautical Information Publication
AIS	Aeronautical Information Service
AUP	Airspace Use Plan
EASA	European Aviation Safety Agency
EFB	Electronic Flight Bag
FAA	Federal Aviation Administration
METAR	Meteorological Terminal Aviation Routine Weather Report
NOTAM	Notice to Airmen
SIGMET	Significant Meteorological Information
ANS	Air Navigation Services of Czech Republic
TAF	Terminal Aerodrome Forecast
VFR	Visual Flight Rules

Chapter 1

Introduction

This thesis reviews the standards, types and the use of an *Electronic Flight Bag* (*EFB*) system used in aviation for private and recreational pilot considering the European aviation standards and regulations. The actual implementation of the *EFB* is designed to be operated on hand-held devices running Android operating system.

The main goal of this thesis is to create an *EFB* for Android OS based on the European standards and regulations that can be used by pilots in General Aviation. The application is designed to help the flight crew during the preparation and the actual flight by providing helpful information and intuitive user interface. Further it should assist the pilot with navigation, planing, performance, balance and weight calculations. The application was purposely designed to be used in General Aviation some of its features were downscaled and migrated from the *EFB* applications used in commercial aviation.

This document is divided into six main chapters. The first chapter (2) discusses the standard and official regulations of the use of an *EFB* and provides a brief look into the already existing *EFB* applications and their features. This section is followed by the application design description including a short introduction into our chosen platform of implementation in section 3.1 and the actual design of the application in section 3.2. This is followed by the implementation details described in chapter 4. The testing and evaluation of the application is discussed in chapter 5. Chapter 6 concludes this work including some ideas for further development.

Chapter 2

Electronic Flight Bag

An *EFB* is a device that helps the pilot and the flight crew to complete tasks easily and more efficiently. The name of the device can be traced back to the aviation bag which contains all manuals maps and other flight related documents used by the crew during the flight. The aim of an *EFB* is to replace all paper documents with a single electronic device containing all flight relevant information. The main and the most obvious reason to use an *EFB* instead of paper documents is the weight and the form factor.

An *EFB* is used in the commercial aviation officially, regulated by strict rules and certified to show compliance to aviation regulations. Regulations must be kept during the design and implementations process. In General Aviation there is no need for an *EFB* design and development certification, but there are some regulations for their design and use.

2.1 Regulatory Requirements

The above mentioned regulations and rules were specified and released by two main aviation rule making organizations. The first one is the *Federal Aviation Administration*, the *FAA* [32], that regulates the aviation on the US territory. The second is the *European Aviation Safety Agency*, the *(*EASA) [31] with the same authority in European member countries. Both of these organizations have their own published sets of regulations and rules, that differ in the details. The following sections will describe the classification of the *EFB*s introduced by both organizations.

2.1.1 Federal Aviation Administration Requirements

This section is based on the official advisory circular number *AC 120-76C* released by the *FAA* [30]. This advisory circular is dedicated to every commercial aviation operator, who wants to replace paper documents with digital one. It also describes the certification processes and approval to use this kind of electronic documents during the flight.

The *FAA* also released the document *AC 91-78* [26], that specifies the conditions of use of *EFB*s of class 1 and 2. This document was dedicated to the operators of airplanes with piston engines in General Aviation. Based on this regulation, the piston engine airplane operators are able to legally use an *EFB* system during the whole flight in case the *EFB* fills the following requirements:

- The *EFB* is functionally equivalent to the original papers.

- All data are up-to-date and valid.
- All the functions correspond to the application class A or B defined in document *AC 120-76C*

The *EFB* can be used without any further approval in case it fulfills these requirements. This system can also have additional functionality than defined, but then it could not be taken as a legit substitution of the paper documents.

There are two more advisory circulars connected to *EFB*s released by the *FAA*. First is the *AC 20-173* [27] that defines the physical installation of the parts of the *EFB* into the cockpit. The second is the *AC 91.21-1B* [25], describing the use of portable devices aboard aircrafts.

The following specifications are based on the *AC 120-76C* [30].

EFB hardware

Class 1: A portable commercial off-the-self computer with no *FAA* design, production or installation approval is considered to be a portable electronic device Class 1. This device is not mounted to the aircraft and has no connection to the aircraft systems or data, neither is connected to a dedicated aircraft power supply. It can be temporarily connected to an existing aircraft power supply only for battery recharging. A class 1 portable device with type B applications for approach chart, electronic checklist or aeronautical charts must be appropriately secured, and viewable during flight and must not interfere with flight control movement. These kind of portable devices are not considered to be part of the aircraft type design.

Figure 2.1: Example of a Class 1 *EFB*.

Class 2: A Class 2 *EFB* hardware is considered the same kind of portable commercial off-the-self device with no *FAA* approval, just like in case of the Class 1 hardware. The main difference is that the Class 2 device is typically mounted, but must be capable of being easily removed or attached to the mount by the flightcrew personnel without any additional tool. These devices can be connected to the aircraft power, data ports (wired or wireless) in accordance with *AC 20-173* [27]. Class 2 *EFB* may consist of modular components that include all the hardware and software that support *EFB* intended functions. These devices are also not considered to be part of the aircraft type design.

Figure 2.2: Example of a Class 2 *EFB*.

Portable *EFB*s Class 1 and 2 are limited to host Type A and Type B software applications with intended functions only.

Class 3: *EFB*s installed in accordance with applicable airworthiness regulations are considered Class 3 *EFB*s. The devices are built into the aircraft based on regulations defined in document *AC 20-173* [27].

Figure 2.3: *Falcon 5X Bizjet* with built in Class 3 *EFB*.

***EFB* software application**

Type A: Software applications that are paper replacement applications intended for use during flight planning, on the ground, or during noncritical phases of flight. The data provided by the application typically provide static data. Failure condition classification of these applications is considered as minor or nonhazardous. Typical features of a Type A application are:

- flight procedure guides,
- airplane manuals,
- service and Maintenance manuals,

- flight manuals,
- international procedures,
- airplane performance data,
- NOTAMs and AIP (more in chapter 3.2.1),
- etc.

Type B: Software applications that are paper replacement applications that provide the information required to be accessible for each flight at the pilot station. These are primarily intended for use during flight planning and also during all phases of flight. The data provided by Type B software are dynamic. This means the crew can interactively manipulate with them. Failure condition classification of these applications is considered as minor or nonhazardous. Typical Type B application features are:

- airplane performance calculations for flight take-off and landing,
- creating and editing flight plans,
- aircraft weight and balance calculations (further discussed in 2.1.1),
- interactive flight maps,
- maps showing own position (further discussed in 2.1.1),
- electronic checklists for regular and emergency procedures (Defined in *AC 120-64* [24] for Operational Use and Modification of Electronic Checklists),
- weather information,
- flight information,
- etc.

Type C: Type C software application are certified by *FAA* based on *RTCA/DO-178B* [23], or other certification process. These systems also feature non-*EFB* features navigation, communication and surveillance that require *FAA* design, production and installation approval. This software is approved for surface and airborne functions, and is considered as a major hazard based on the failure condition classification.

Weight and Balance Calculation Requirements

The weight and balance calculations have to be based on the existing data from the valid and approved aircraft manuals. All used calculations and algorithms must be verified based on these original documents. Interpolation in calculations are allowed but it is forbidden to create new algorithms to calculate parameters. It is also forbidden to allow to enter input data that are outside a given range defined by the manuals.

In case of a full paper replacement in the cockpit, the use of two different *EFB* systems is required.

Own-Ship Position Source and Display Characteristics

As this feature is a part of Type B software application, it should not create a hazard to the aircraft. Its purpose is only to help orientation, and is not intended for navigation. It is recommended to use its own built-in or external Global Navigation Satellite System (GNSS) source. The own-ship position is allowed to be visible only if the GNSS source accuracy is less then 40 meters. Showing the position on the airport diagrams is limited to a maximum ground speed of 80 knot.

2.1.2 European Aviation Safety Agency Requirements

This section sums up the system description and classification of *EFB* systems defined by the *AMC 20-25* [28] released by the *EASA*. *AMC 20-25* applies to all commercial air transport operators.

Based on the *AMC 20-25* specification, the paper documents can be fully replaced with an *EFB* only after a detailed *EFB* risk assessment and simulated flight training sessions that verify the use of the *EFB* under normal and emergency conditions.

These systems can be categorized based on their hardware and software by the following specifications described in *AMC 20-25* [28].

EFB systems hardware

Portable *EFB*: A portable *EFB* is hosted by a portable platform that is not an integral part of a certified aircraft configuration. A portable device can be operated inside or outside an aircraft and has no limitations in size and weight unless its presence does not compromise flight safety. Powering of the device is to be provided through a certified power source. In case the portable *EFB* is installed into the cockpit, it has to be easily removed without the utilization any additional tools. The conditions for use of its transmitting capability must be kept within the limits specified in an approved Aircraft Flight Manual. In absence of these information the transmitting capability of the *EFB* system is allowed to be used only during non-critical phases of flight. Portable *EFB* systems are allowed to be used during all phases of flight if secured to a certified mount in a way that allows its normal use. A portable *EFB* may host a Type A and/or Type B software application and also non-*EFB* software applications.

Installed *EFB*: An installed *EFB* is considered to be a part of an aircraft, covered with the aircraft airworthiness approval. It is managed under the aircraft type design configuration. Besides the certified application it can host Type A or Type B software applications that do not affect the operation of the certified one.

Software applications for *EFB* systems

Type A: A Type A applications malfunction or misuse have no safety effect. These applications can be hosted on both portable and installed *EFB*s and do not require any approval. Examples of a typical content of a Type A application are:

- digital copies of aircraft certificates,
- maintenance manuals and aircraft parts manuals,
- other documents and forms required to be carried,
- airport diversion policy guidance,
- trip schedule and passenger informations,
- interactive crew rest calculation,
- interactive reporting forms.

Type B: A Type B software application malfunction or misuse is considered as a minor failure condition and they are not considered as a substitute nor duplicate of any system and function required by the operational rules and airworthiness regulations.

Type B applications do not require airworthiness approval but some features like own-ship position displaying and weight and balance calculations require detailed documented evaluation. Some other of the typical Type B application features are:

- the aircraft Flight Manual, Operations Manual, and Operational Flight Plan,
- aircraft airworthiness records,
- meteorological information with graph interpretation,
- NOTAMs and AIS,
- aeronautical charts and airport surface maps,
- aircraft performance calculations.

Performance Calculations

Input and output data must be visually separated in an *EFB* application. All information must be easy to read with all the units visible. In case of the input data all input have to clearly present, whether it is from a default, imported or manually entered source. The system has to make a clear visual notification case of not accepted input values. All values should be independently modifiable and the result must be recalculated instantly for every change of the input value.

Weight and Balance Calculations

These calculations must be based on the actual values published in the flight manuals. Calculations may use algorithms or tables for calculations, also with data interpolation but may not extrapolate outside of the published interval. The calculation algorithms and data tables validity must be confirmed.

Airport Moving Map Displays With Own-ship Position

Own-ship position on moving maps is not used as the primary means of navigation and it should be used only in conjunction with other materials and procedures. Its malfunction is considered as minor safety issue. The displayed position accuracy may not exceed 50 meters.

2.2 State-of-the-art Applications

There are numerous applications for document management applications that can be used to fulfill the requirements for a Type A *EFB* both by the FAA (2.1.1) and EASA (2.1.2) definition. Therefore, here we focus on applications that implement some features of the Type B applications with navigation, flight planing, and various calculation functions.

All the here listed applications were selected based on their high download count and rating, and their features that can give inspiration for the future application design (chapter 3.2).

All the bellow mentioned data are valid to the application versions latest version as of May 6th, 2017.

2.2.1 State-of-the-art Applications For Android

This chapter gives a short description about the most used free and free to try *EFB* and airplane navigation applications on the *Android Play Store* [9].

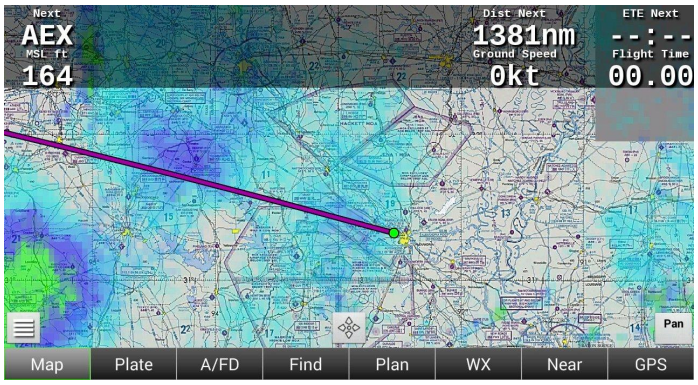

Figure 2.4: *Avare* app screen.

Avare

Avare (figure 2.4) is the most used free flying navigation application made for the US and Canadian pilots. It provides up to date offline moving map and all official *FAA* published data with detailed airport and terrain diagrams. The user can create own lists and flight plans, and add own drawings and comments on the map.

Based on the user reviews this application has all advanced features a pilot needs for easy navigation. Based on some reviews it has a rather non intuitive user interface that needs a longer learning-curve to get used to.

Figure 2.5: *Flight Assistant v2* home screen.

Flight Assistant

Flight Assistant (figure 2.5) is a solid Type B *EFB* application with high detailed offline vector maps covering the global aerospace. It provides the official AIP, Weather, NOTAM, TAF and METAR reports. Furthermore it provides airplane weight, balance and perfor-

mance calculations. The user interface of the application is straightforward and easy to use.

Naviator

Naviator (figure 2.6) is a 30 day trial application designed mainly for navigation purposes with flight recording and tracking. The application features a 3D map view and easy planning. Weather, NOTAM and METAR informations are available for worldwide global coverage, but the AIP and airport charts are only provided for Canada and the US regions. The application provides a simple user interface and a set of useful functions.

RWY

RWY (figure 2.7) is a 30 day trial application that was designed for Australian pilots but since the latest versions it has a global support. Besides the easy to use planing it features an up-to-date AIP download for every country, and simple METAR, NOTAM and TAF browsing.

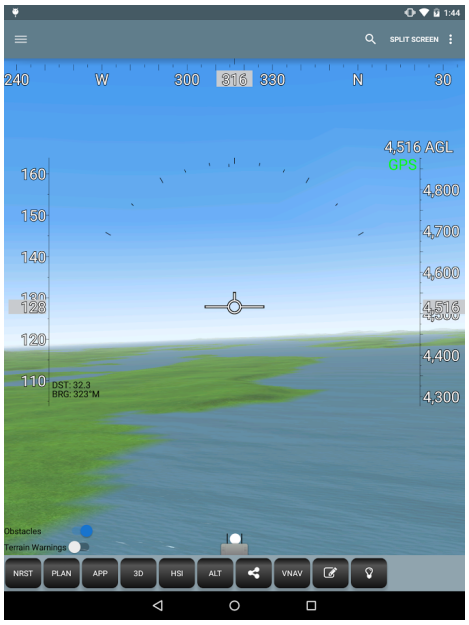

Figure 2.6: *Naviator 3D* navigation screen.

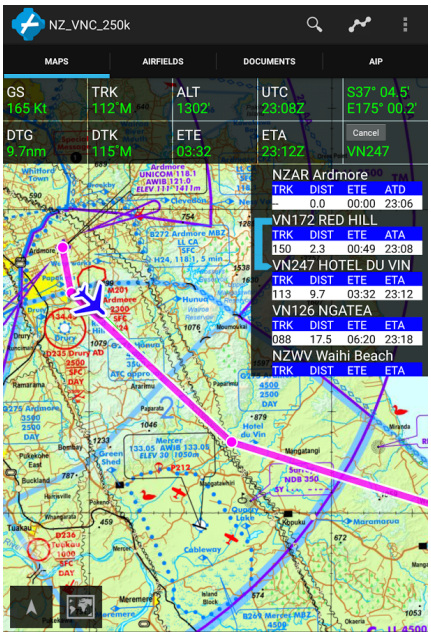

Figure 2.7: *RWY* navigation view.

SkyDemon

SkyDeamon (figure 2.8) is Europe's most popular flight planning and in-flight navigation application. It features high detail aeronautical charts, provides weather, NOTAM and METAR informations and airplane weight and balance calculations. The most unique feature of this application is the clear virtual radar for airspace, terrain and obstacles display.

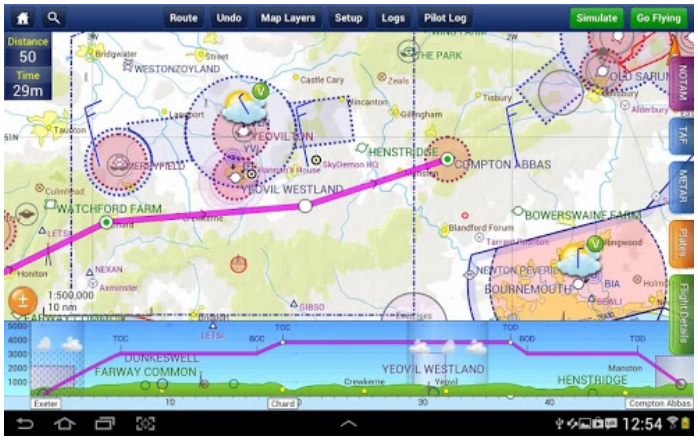

Figure 2.8: *SkyDeamon* screenshot.

2.2.2 State-of-the-art Applications For iOS

The following applications are some of the most used *EFB* and flight navigation applications amongst the many, accessible in the *Apple App Store*[11] for an *iOS* device.

OzRunways

OzRunways (figure 2.9) os the iOS version of the RWY (section 2.2.1) Android application. It provides however some additional features to it, like elevation graphs and weather map overlays.

The biggest benefit of the OzRunways is the straightforward user interface. It has a 30 day trial period, just like its Android version.

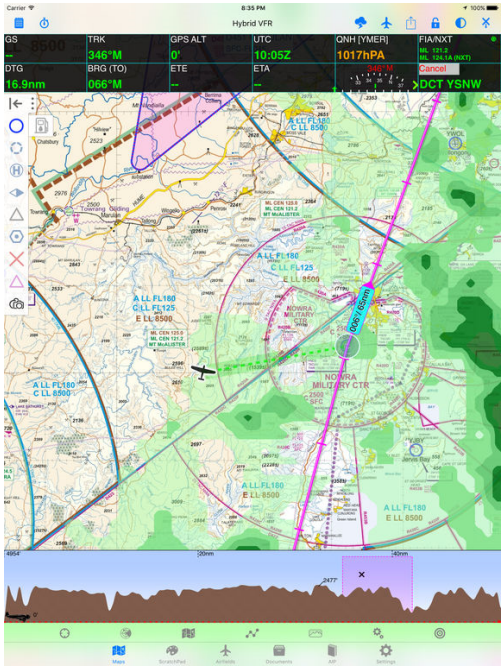

Figure 2.9: *OzRunways* navigation screen.

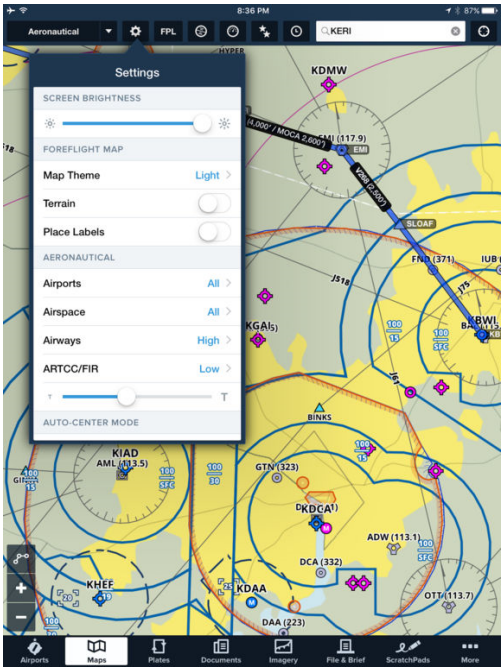

Figure 2.10: *ForeFlight* map view.

ForeFlight

ForeFlight (figure 2.10) is an award winning flight planning and navigation application. It features easy flight planning, detailed weather reports, weight and balance calculations, and up-to-date approach and taxi charts.

The most unique feature amongst the other application is the synthetic terrain overlay, that provides map color-coding based on the terrain elevation and flight altitudes. The application has a 30 day trial with different time limited subscriptions and additionally purchasable features.

Honeywell MyGDC

Honeywell's *MyGDC* application provides access to their global flight services database and needs *Honeywell*'s *Global Data Center* access to be used. It is a complex solution for for managing flights with automatic data synchronization and high quality up to date maps and informations.

Figure 2.11: *Honeywell MyGDC.*

Chapter 3

Design of an Electronic Flight Bag for Android

This chapter defines the design of and *Electronic Flight Bag* for the *Android* operation system. It describes the chosen operating system in section 3.1 before the actual application design discussed in section 3.2.

3.1 Android Operating System

Android is the most popular mobile OS for phones, watches, cars, TVs and other digital platforms [2]. Due to it's popularity the author is going to use the Android OS as a base platform for its *EFB* application. The following sections are based on the articles [6], [10] and the official developers website of Android OS [5].

3.1.1 Historical Overview

Android, Inc. was founded in Palo Alto, California in 2003 with the goal to create a multi-platform, smart mobile OS. The company was bought by Google in 2005 and released the first version of the OS in 2008. The latest, version of the OS *7.1*, codename *Nougat* was released in *October 4, 2016*. The list of all versions of the OS is listed in table 3.1.

Since its first release the OS remained OpenSource, allowing the manufacturers to customize and modify the system. The *Android Open Source Project* (*AOSP*) source code is free to use for developers to create their own modified version of the OS. The most well known is the *CyanogenMod* which opens features not available in the official Android releases, like CPU clocking and other hardware related settings and options.

As of January, 2017 the oldest still supported version is *Lollipop*. The actual API distribution amongst the users can be followed on the official developers dashboards website [4]. The data are collected every week based on the *Google Play* store usage. This helps the developers to select the minimal API version for development so most of the actually used devices will be able to run the new application.

Version number	Code name	Initial Release Date	API level
1.0		23.09.2008	1
1.1		09.02.2009	2
1.5	Cupcace	27.04.2009	3
1.6	Dounut	15.09.2009	4
2.0-2.1	Eclair	26.10.2009	5-7
2.2-2.2.3	Froyo	20.05.2010	8
2.3-2.3.7	Gingerbread	06.12.2010	9-10
3.0-3.2.6	Honeycomb	22.02.2011	11-13
4.0-4.0.4	Ice Cream Sandwich	08.11.2011	14-15
4.1-4.3.1	Jelly Bean	09.07.2012	16-18
4.4-4.4.4	KitKat	31.10.2013	19
5.0-5.1.1	Lollipop	12.11.2014	21-22
6.0-6.0.1	Marshmallow	05.10.2015	23
7.0-7.1	Nougat	22.08.2016	25-25

Table 3.1: List of Android OS version history.

3.1.2 System Features

Interface

The default user interface is based on direct manipulation using touch inputs based on real-world actions like tapping swiping and pinching. The platform also allows to use on-screen virtual or real life keyboards and game controllers via *Bluetooth* or *USB*. Internal sensors such as accelerometers, gyroscopes and proximity sensors can also respond to user actions.

The graphical user interface main screen, the '*hub*' is the Android analogous for the desktop, found on PCs. The home-screen (figure 3.1 and 3.2) of the hub, is typically made of application icons and interactive application widgets. The home-screen is heavily and easily customizable with new pages and widgets. The feel and the looks are customizable using third-party themes and launchers available on the *Google Play* store.

Another main component of the GUI is the notification bar on the top of the screen that contains all important information such as time, battery and network status. The user reaches different application notifications by expanding (swiping) down this bar.

Applications

Applications (apps) extend the features and functions of the OS. Apps are written using the *Android Software Development Kit* using *JAVA*, that may be combined with *C/C++*.

The *Android Play* store is a built in application in the OS that allows to download, install, update and remove *Android Application Package*s (*APK*s). This store contains millions of third-party application for free or to purchase with different online payment methods. Besides of the official *Play* store, there are also third-party application marketplaces like the *Amazon Appstore*, *GetJar* and *F-Droid*.

Memory Management

Since Android was designed to be used on battery powered devices, power and memory management is an important part of the OS. Every currently not used application is sus-

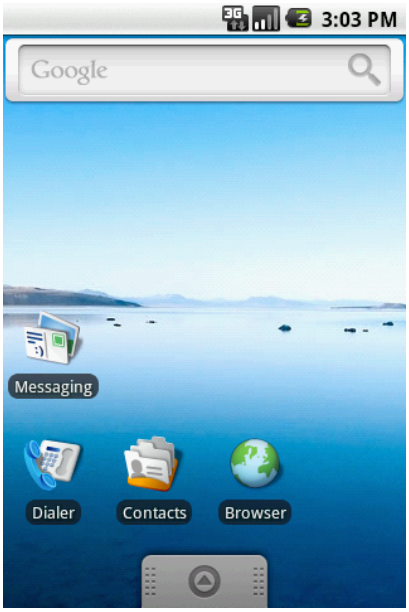

Figure 3.1: Homescreen of *Android* 1.6.

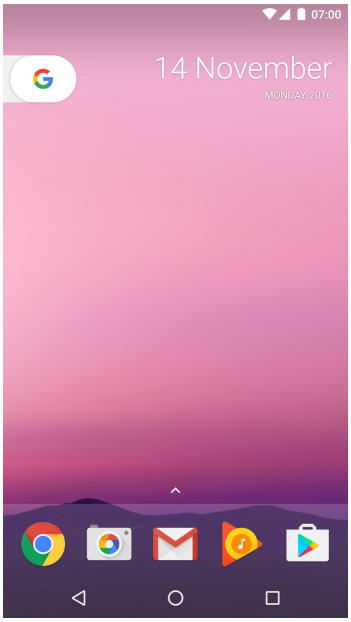

Figure 3.2: Homescreen of *Android* 7.1.

pended while it still allows immediate use. The Running applications are stored in the memory until closed. When the memory runs low, the longest inactive application gets closed. This memory management is clearly visible on the application (*Activity*) life-cycle that is discussed in chapter 3.1.4.

Virtual Reality

Google has announced its Virtual Reality platform, the *Daydream* in *May, 2016*. The platform is now integrated into the latest *Nougat* version of the OS.

3.1.3 Platform Architecture

The Android OS is based on the Linux kernel long-term support branches. The used Linux kernel is modified by Google, and completed by components like the *Binder*, *Logger*, power management features and out-of-memory handling.

The flash storage is divided into several partitions like the `/system`, for the OS itself, and `/data`, for the user data application installations. The owners are not given *root* access to the OS by default. However by exploiting the security flaws of the system or by using a rooted build of the OS the user can access these data. Rooting a device is not recommended, because it makes the system more open for viruses and malwares.

The figure 3.3 shows the major components of the Android platform [8].

Linux Kernel

The base is the above mentioned Linux kernel. This allows android to take advantage of key security features like filesystem permissions, cryptography, user security features, device administration and process isolation. The kernel also gives the base for hardware driver development for different devices.

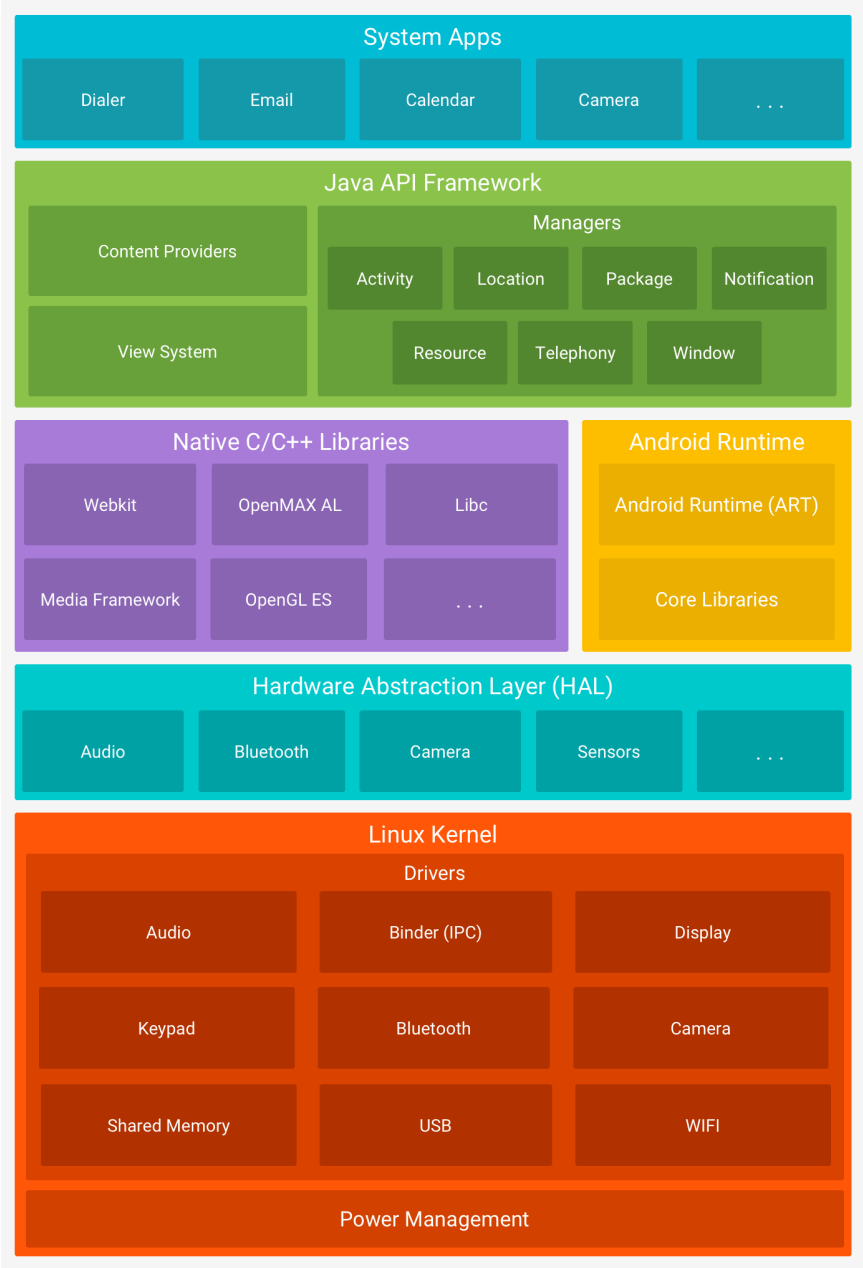

Figure 3.3: *Android* software stack.

Hardware Abstraction Layer

This layer provides standard interface that connects the hardware capabilities to the higher-level *Java API framework*. This layer contains multiple library modules, which implement interface to specific hardware, such as the camera, display, speaker or different sensors.

Android Runtime

Prior Android 5.0 (API level 21), *Dalvik* was the Android runtime, and only one instance of it was running. Since then, newer Android versions use *ART*. *ART* is written to run on low memory device in multiple virtual machines It executes special bytecode format (*DEX*), that is optimized for minimal memory footprint. That means that every application runs on its own instance of the ART. Some of the major features o *ART* include the following:

- Ahead-of-time (AOT) and just-in-time (JIT) compilation.
- Garbage collection (GC).
- Debugging support with detailed diagnostics, crash reporting and field monitoring.

Native *C/C++* Libraries

The Android platform provides *Java framework APIs* to some native *C/C++* libraries like *OpenGL* or *SQLite*. The *Android NDK* (*Native Development Kit*) is used to access these native platform libraries from a *C/C++* code.

Java API Framework

The entire feature-set of the OS, is available through a set *Java APIs*. An application can be built by reusing these modular core components like:

- **View System** to build user interfaces including buttons, list, and other elements.
- **Resource Manager** provide access to non-code resources like localized strings, graphics, and layout descriptions.
- **Notification Manager** enables the application to display different alerts and notifications.
- **Activity Manager** that manages the life-cycle of the application. (more in section 3.1.4)
- **Content Providers** enable to access and share data with other applications.

System Apps

Android has some built-in core apps for messaging, internet browsing, calendar, contacts and more. These application can be replaced by third-party applications. Developers can use these applications by invoking them inside their application so no further functional development is needed.

3.1.4 Application Components

This section is based on the official Android API Guides [3]. It describes the main application components used during a typical application development.

App Manifest

The `AndroidManifest.xml` is a mandatory file located in the root directory of every application. This file provides all the essential information about the App, which the system must have before it can run the application. The manifest `.xml` file contains the following information:

- The name of the Java package of the application. This name serves as a unique identifier for the application.
- It describes the components of the application, such as activities (3.1.4), services (3.1.4), intents (3.1.4) and content providers. Each of these components name and class must be declared here, so the system can run them.
- Each application needs permission to access some protected parts the API. These are for example to use the storage, connect to the internet, or access data from sensors. These access permissions must be declared here.
- The minimum level of API that the application requires.
- List of the linked libraries to the application
- The application version, that helps during application updates from the application store.

App Resources

Every resource, like strings and images should be externalized from the java code for better maintainability. This allows to provide alternative resources for specific device configurations such as languages or screen sizes. These resources should be well organized in the `res/` directory of the application. The resources are accessible in java via their the automatically generated `R` class with the syntax `[<package_name>.]R.<resource_type>.<resource_name>` or with `@[<package_name>:]<resource_type>/<resource_name>` from an other `.xml` resource. The most commonly used resources are the following:

- Colors - used to assign name to color values defined by #RGB, #ARGB, #RRGGBB or #AARRGGBB hexadecimal form.
- Drawable - used to define graphics like bitmaps, shapes, nine-patches, layers or transitions between other drawable.
- Layout - definition of the UI architecture for window and other UI components.
- Menu - defines application menu contents and the menu items behavior.
- String - provides text strings in different defined languages.
- Style - Defines the format and the basic, general look of the UI.

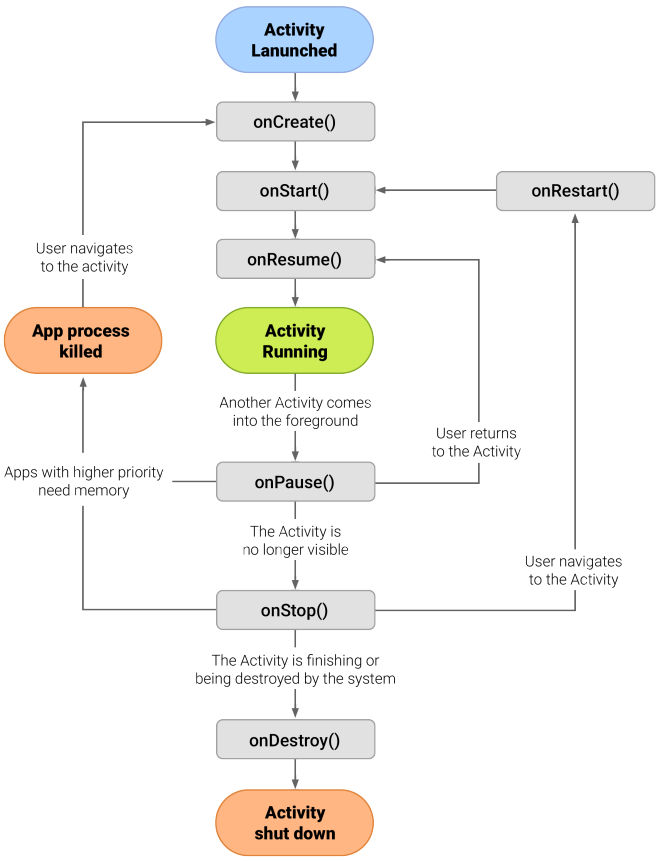

Figure 3.4: The lifecycle of an activity.

Activity

Activity is the basic component of the application that provides a screen, window for user interaction. Each time a new activity is started, the previous is stopped and preserved by the system on the activity stack. These state changes are notified to the activity via the lifecycle callback methods like `onCreate()`, `onStart()`, `onPause()` or `onStop()`. The full activity lifecycle is visible on the figure 3.4.

The two most important to implement methods of this lifecycle are:

- `onCreate()` - This method must be implemented by the developer. This creates the activity and initializes the essential components and the layout of the user interface.
- `onPause()` - This method should be used to commit any changes that should be persisted for further sessions and starts.

Fragment

Fragments represent the behavior or one part of the user interface of the activity. One activity can combine more fragments and change between them. Fragments have their

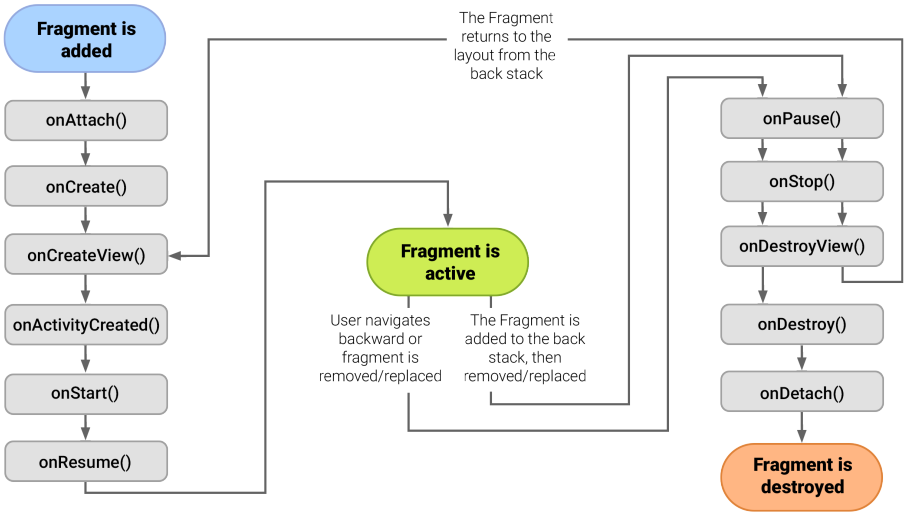

Figure 3.5: The lifecycle of a fragment.

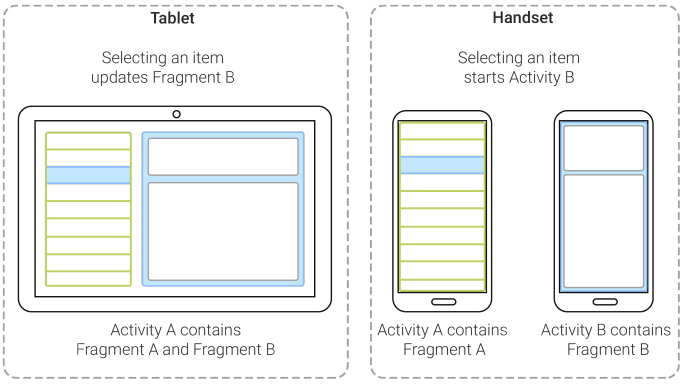

Figure 3.6: Example of using fragments for different size devices.

own life-cycle (figure 3.5) with own events that are directly affected by the host activity's lifecycle. The advantage of using fragments is the re-usability of the different UI components together with their behavior.

The idea of fragments come from the UI design point of view. By dividing the layout into more sub-components it is much easier to modify the application appearance at runtime. It also enables for tablets (bigger screen devices) to show more components the same time, while less on the smaller-screen devices. This principle is visualized on figure 3.6.

The following methods are recommended to implement while using fragments:

- `onCreate()` - This method is called when creating a fragment, and should be used for the essential component initialization.
- `onCreateView()` - This method is called when the component is drawn for the first time. This method should implement the user interface initialization.
- `onPause()` - Here should be the implementation to commit any changes that should be presisted beyond the current session.

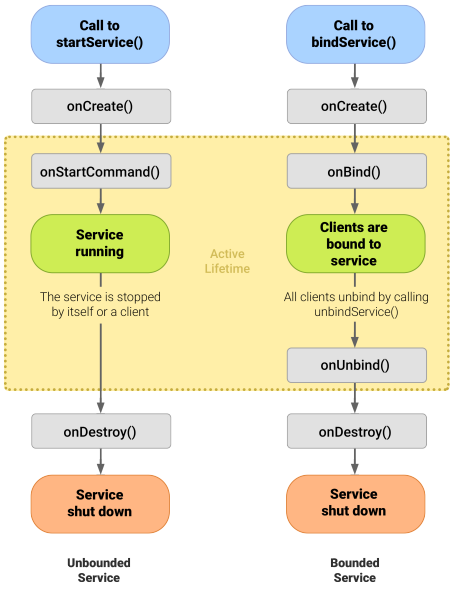

Figure 3.7: The lifecycle of the unbounded and the bounded service.

Service

Services are application components that perform operations in the background without user interface. Any other application component can bind to the service to interact with it or to perform interprocess communication.

There are three different types of services:

- Bound - This service offers a server-client interface. This allows the different application components to interact with the service, send requests and recieve result. The service runs as long as an application component is bound to it. A service can be bound to multiple components at once, but when all of them unbind, the service is destroyed. Starting a binded service is done by calling the `bindServeice()` method of the application component.
- Started (Unbound) - A started service can run in the background indefinitely, even after the component that started it is destroyed. It is usually used to perform single operations that does not return any result to the caller. After the operation is finished, the service stops itself. This type of service is started by calling the `startService()` method of the application component.
- Scheduled - The newest type of service introduced in Android 5.0 (API level 21). Using `JobSheduler` any kind of service can be registered to run at a given time. The Android system automatically schedules and performs the tasks at the given time.

The figure 3.7 shows the lifecycles of both bound and unbound services.

Intents

Intents are messaging objects, used to request actions from another application and components. They can be used to:

- Start an Activity - An intent can be passed to the (startActivity()) method. The intent describes the activity and carries any necessary data for the activity to be started. The result of the Activity is also returned as an Intent object that can be handled by the `onActivityResult()` callback.
- Start a Service - The same goes for Services as for the Activities. An intent can be passed to the `startService()` method. Intents are sent to services via `bindService()` in case of further client-server-like communication.
- Delivering and broadcasting - Broadcast and other types of messages to other applications are possible by passing an intent to the `sendBroadcast()` method.

There are two types of intents that can be used:

- Explicit intents specify the component by the full class name. These intents are used inside an application to start the component of an application.
- Implicit intents do not specify a specific component, but declare a general action to be performed and allows for an other application to handle it. This can be for example to send a text message or to show position on the map.

Other components worth to mention

- `Dialog` - A dialog is a window that appears on top of the running application window. Normally dialogs do not fill the screen, but they can be implemented in a full screen mode with custom layouts. Dialogs are usually used when the application needs an additional information to be filled by the user.
- `View` - This class represents the basic building block of the user interface components. It occupies a rectangular area on the screen and it is responsible for drawing and event handling. The most known and used subclasses are `TextView`, `Button`, `ListView` or `ImageView`, but it has over 100 subclasses implemented in the Android SDK for different purposes.
- `SharedPreferences` - An interface to access and modify preferences and settings for the application. It holds key value structure for saving the requested data. It is used to recall settings that were saved during previous application uses.
- `Notifications` - A notification is a message that is displayed outside of the application window usually on the system notification bar. The notifications are used to inform the user about something important, usually about an ongoing background process or about incoming messages and events.

3.2 Application Design

Before the actual design, it is necessary to define the goals and the target user group of the application. These aspects have a major impact on the final application. Furthermore it is also necessary to design an application that isn't just a copy of the already existing applications.

Based on the defined *EASA EFB* usage rules (2.1.2), a non-certified software can be used only in sport and recreational flying. Therefore our application will target this segment of aviation. Based on chapter 2.2, we can say that the majority of the applications mainly focus on the US airspace, for the European use they lack some features. Due to these reasons our target is to create an application designed to be used in sport and recreational flying focused on Central European regions.

3.2.1 Features

The following features were defined based on the *FAA* and *EASA*s *EFB* rules, and on the earlier mentioned applications user reviews and own user experience.

As pilots are gathering informations from different paper manuals and guidelines, the first main function of the developed application is to provide all these informations in one, well structured way. A separate document browser for the airplane manuals and for the official AIP and VFR documents, released by every country's air navigation service provider.

A second useful feature is the airplane checklists and procedures manager. These are used in different phases of flight and must be strictly followed. It should allow the user to create and edit the different lists, and to create a different set for different airplanes he or she uses.

Another useful functions are different unit conversions and the airplane weight, balance, performance calculations. These calculations are necessary to be executed for every flight due to safety reasons. A well designed easy to use and precise calculator can be a big help for the crew.

Some of the most important data during the flight are the different *NOTAM*, *TAF* and *AIP* informations that are released by the air navigation service providers and give information about the actual status, occupancy and rules of a given airspace. At the same time the *METAR* provides weather reports for a given area. All these reports are released in different time intervals so these features need constant Internet connection to be up-to-date.

The last feature our application should provide is a simple to use and helpful navigation that allows the pilot to create flight plans and routes he wants to follow during flight.

3.2.2 User Interface

The user interface and the whole user experience is one of the most important parts of a useful application. In case of an *EFB* it is even more essential. The pilot has no time to search for some data on the screen for too long. Everything has to be non confusing and clear so it does not interrupt the pilot from the actual piloting.

During the visual design of an *Android* application it is recommended to follow the official *Material Design* guidelines [18] created by *Google*. This design is the part of the whole android system since the *Android 5.0 (API 21)* so the users are familiar with it. The created application will be more user friendly and easy to use following this refined design.

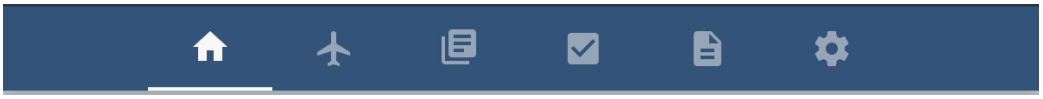

Figure 3.8: Main tabbed menu.

The basic idea is to divide the main features into different tabs (figure 3.8). The arrangement of the tabs is based on logical order based on significance. The first screen to show up is the map and navigation screen where the pilot is informed if all app data are ready and up-to-date to be used. The second tab is for the airplane weight, balance and performance settings and calculations before the actual flight. The third is for the official *AIP* and *UUP* documents followed by the checklists and the documents views. The last tab of the application is designed for basic application settings and data (map, *AIP*, *UUP*, ...) updates.

Chapter 4

Application Implementation

The created *EFB* application is named *EasyFlightBag* with icon showed in figure 4.1, designed based on the Material Design [18] icon design standards. The minimal Android API support was defined to support Android 5.0 (API 21). The application needs location and file storage read and write permissions for full functionality.

Figure 4.1: *EasyFlightBag* application icon.

When the application is started for the first time, it creates a file structure in the internal storage into the `/EasyFlightBag` folder. The application will further work with the files and data stored in this folder.

4.1 Design Features

The following sections describe all the features and the usage of the implemented *EasyFlightBag* application.

4.1.1 Weight and Balance Calculation

Weight and balance (W&B) calculations are an essential step before flying light aircrafts, as they are sensitive to the different weight distributions.

The user needs to define an airplane before the actual W&B calculation. This is possible by clicking on the airplane icon on the toolbar of the W&B calculation screen (figure 4.4). Then the *Manage Airplanes* dialog (figure 4.2) pops out that allows to add, edit and remove airplane profiles. By adding a new airplane, or by clicking on the edit icon, an airplane editor dialog is showed (figure 4.3). The user is required to fill the form with the valid data based on the airplane manuals. Some of the most significant data here are for example the cruise speed, the empty arm and weight, fuel tank capacities, and the center of gravity limits. These values are used to calculate the W&B before the flight.

After the airplane is selected, the user is requested to fill the form visible in figure 4.4. The flight time and the fuel levels are already filled in if there is an ongoing flight (see

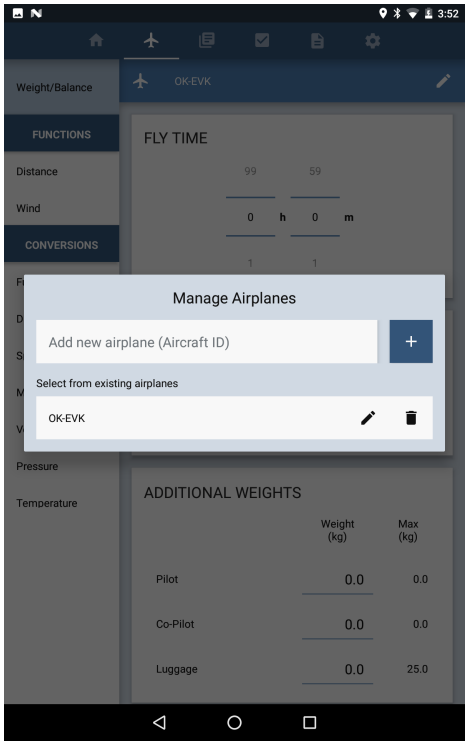

Figure 4.2: Airplane selection dialog.

Figure 4.3: Airplane editor.

Figure 4.4: W&B calculation form.

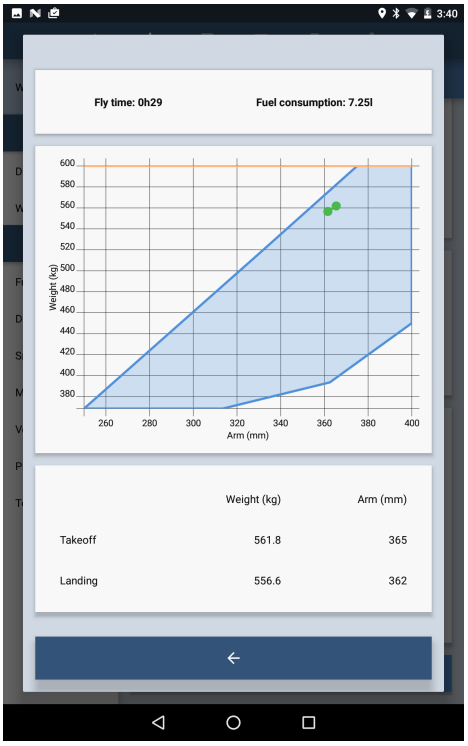

Figure 4.5: W&B calculation result.

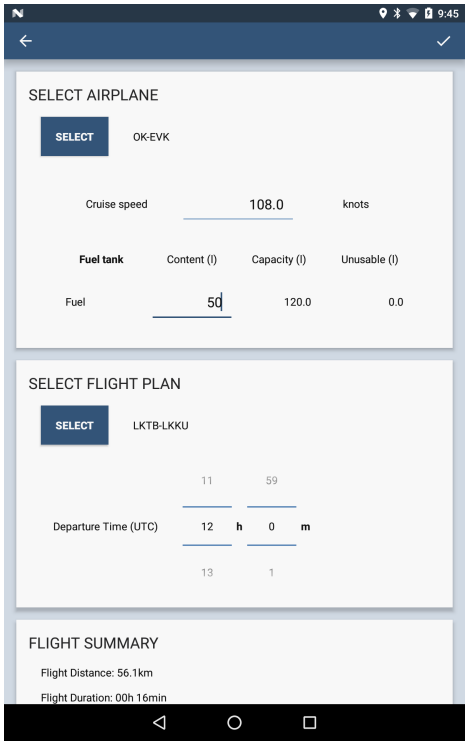

Figure 4.6: Flight planning.

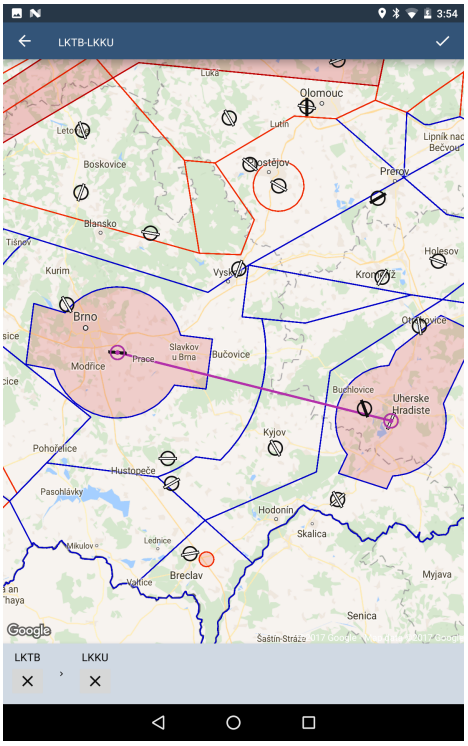

Figure 4.7: Flight plan editor.

sections 4.1.2 and 4.1.3), and it is not possible to change them. It is neither possible to change the selected airplane in this case. By clicking on the confirmation button on the bottom of the form, the W&B is being calculated and the result is shown in a dialog (figure 4.5). This screen shows the center of gravity plot and warns if the airplane is overloaded or if there is not enough fuel to complete the flight.

The airplane properties are individually saved into a *Json* file within the `/EasyFlightBag/Airplanes/` folder. The handling of these files is done by the `AirplaneManager` object, that stores, reads and writes airplane data. The last used airplane is saved to the `SharedPreferences` so it gets preselected on the next application run.

4.1.2 Flight Planing

The flight plan creator window is visible in figure 4.6. There are a few information a user needs to fill. First is to select an airplane (see. section 4.1.1 for more details). The second it the route or the actual flight plan. The management of the flight plans works the same way as the management of the airplanes.

The application helps to create flight plans by plotting on a map (figure 4.7). The way-points of the plan are added by selecting on the map. If an airport was selected, the way-point gets the airport identification. In other case the points are named after their number. The list of the way-points is shown on the bottom list that also allows to remove the points individually.

The flight plans are saved individually into *Json* files within the `/EasyFlightBag/Plans/` folder. A flight plan is managed by a `FlightPlanManager` object. This object is used to read, save and hold the flight plan information.

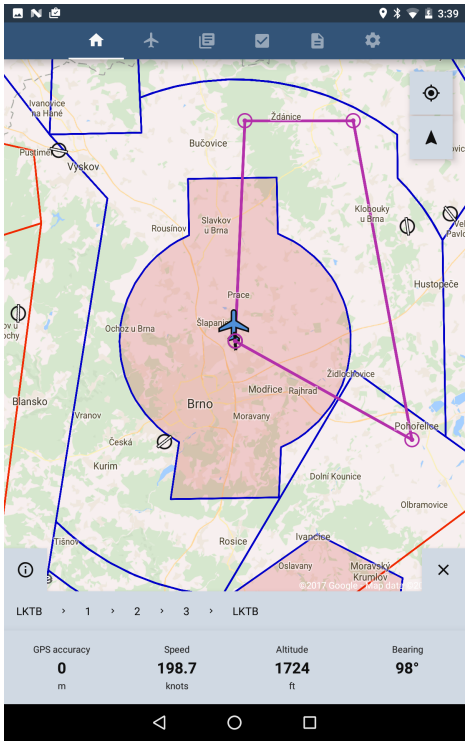

Figure 4.8: Flight navigation screen.

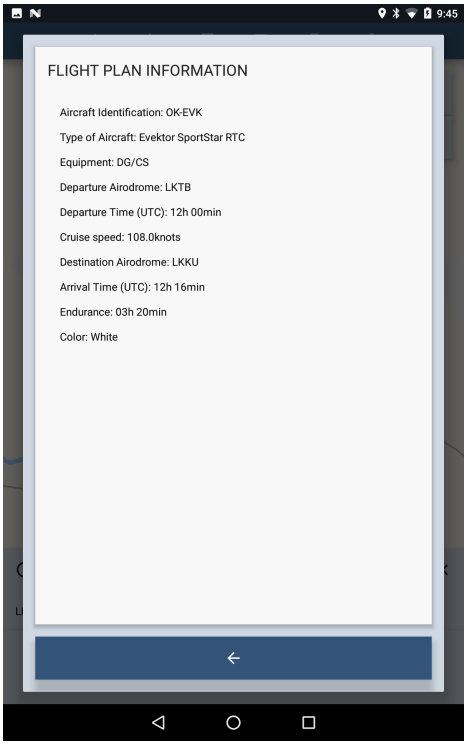

Figure 4.9: Flight details dialog.

4.1.3 Navigation

The home screen of the application is dedicated for the navigation (figure 4.8). The bottom panel shows the most basic information provided by the *GPS* like the accuracy, speed, altitude and bearing. The speed can be switched between the *km/h* and *knots*, while the altitude can be shown in *meters* and *feet*, just by clicking on the value. In case of a *GPS* signal loss, the user is notified with a warning.

The map base is provided by an inbuilt *GoogleMaps* service. The map overlays are provided by the `openAIP` database [20]. It is possible to update the airport and airspace details in the `Settings` of the application (further described in section 4.2). The map overlays are supported for the following countries: Austria, Czech Republic, Hungary, Poland and Slovakia. It is possible to select only one country overlays to be shown or even to turn on all, however it is recommended to select a specific country for faster map loadings. In the `Settings` the user is also able to filter which airport types are showed on the map.

On the top right corner of the map there are two toggle buttons to manipulate with the map view. The top one turns on the follow mode after the user was browsing the map by swiping and zooming. In this mode the map is centered on the own position and follows the movement of the aircraft. The bottom button is used to switch between the *north-up* and the *track-up* map display modes.

The ongoing flight plan is shown on the map with magenta colored markers, and the route way-points are listed under the map view. By clicking on the the information button at this list, the flight plan detail dialog is opened (figure 4.9).

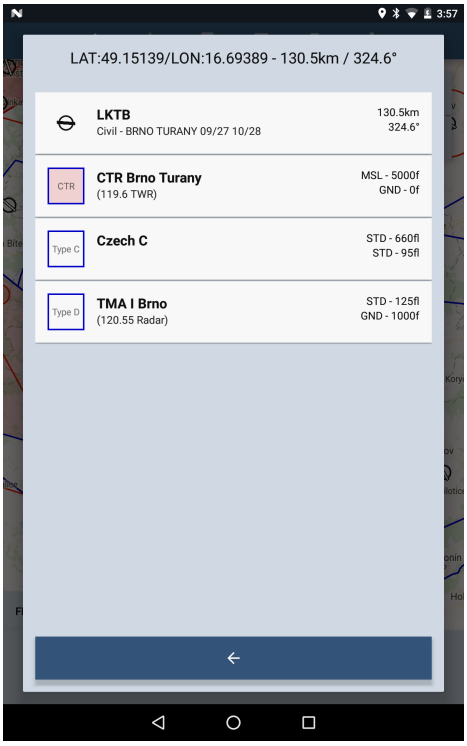

Figure 4.10: Location detail information.

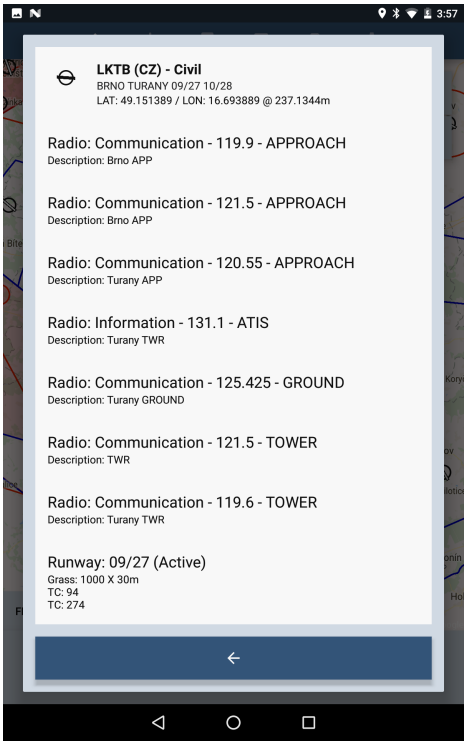

Figure 4.11: Airport details.

Airport and Airspace Details

A detailed view about the airspace and airport details can be opened by clicking on the map or on the flight route way-point in the list. This detailed view is shown in figure 4.10. The list shows the nearby airports of a the clicked location with the additional distance and track from the actual position. The airspaces in the list are the ones that are present at the current location. Their bottom and top elevation limits are also shown here.

By clicking on an airport in the list, the airport details show up (figure 4.11). Here are listed all the known informations about the airport such as it's location, radio frequencies and runway informations. The icon of every airport is generated based on it's type, runway surface and runway heading.

4.1.4 Checklists

The checklist screen is divided into two columns with two different states. One for the airplane and checklist management and one for the checklist execution (figure 4.12). The left column is a list of the existing checklists of a selected airplane that can be changed by clicking on the airplane icon. The header of this left column allows the user to create and remove airplanes and to manage list for these airplanes. By selecting a checklist form the list, the layout changes to the checklist execution layout (figure 4.13) where the user can go through a list step-by-step using the green floating tick button on the right bottom of the screen.

The user can edit the list by clicking on the edit button in the left column header. This opens the checklist editor dialog, where he can modify the list. Every task must be written into an new line so the application recognizes it as an individual task in the list.

Figure 4.12: Checklists browser screen.

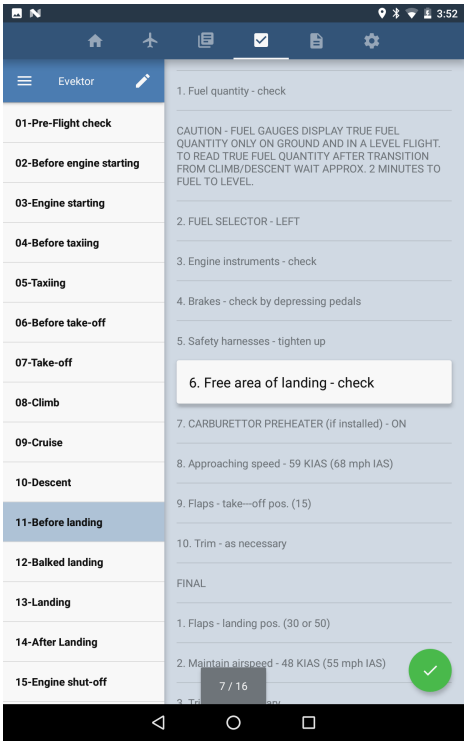

Figure 4.13: An open checklists screen.

Checklists are filled with a quick guide tutorial when the application is started for the first time, so the user gets a quick guide about the usage of this feature.

4.1.5 Aeronautical Information Publications and Document Browser

The *AIP* data are updated manually from the *Settings* tab of the application, where also the last update time is showed. The documents are downloaded from the website of the official *AIP* provider for Czech Republic [1] with a service running individually in the background. The `.pdf` documents are stored in the `/EasyFlightBag/AIP/cz/` folder with a created `data.txt` file, containing the *AIP* folder structure.

The data are structured into multiple column lists on the AIP browser screen (figure 4.14) . The user can search for the documents in a simple side scrollable view. Every new folder layer of the AIP is opened into a new list showing its content.

The application also features a simple `.pdf` document browser visible on figure 4.15. The user can copy his own documents into the `/EasyFlightBag/Documents` folder. The application automatically list all the `.pdf` documents from this folder.

4.1.6 Calculations

Besides the W&B calculation, the application also features unit conversions and some simple and useful functions. First of them is wind speed calculator (figure 4.16). This simple tool is able to calculate the wind speeds from different directions that affects the airplane at take-off, flight and landing. By entering the wind speed, wind direction, and the runway true course (or flight direction) the calculator shows these values in an easy to read diagram.

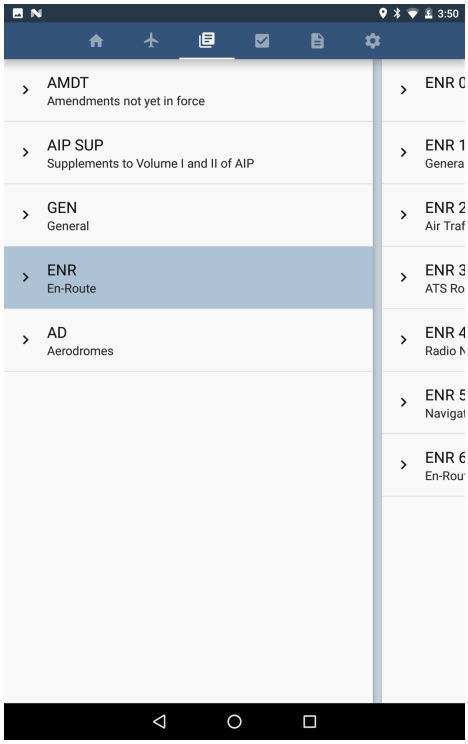

Figure 4.14: *AIP* browser.

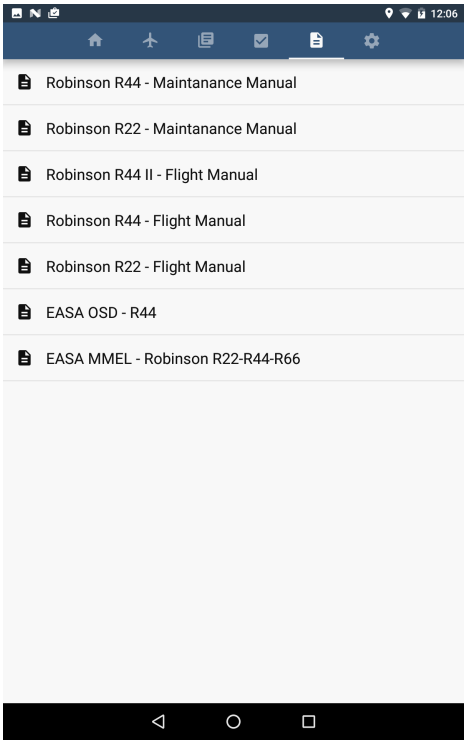

Figure 4.15: Document browser.

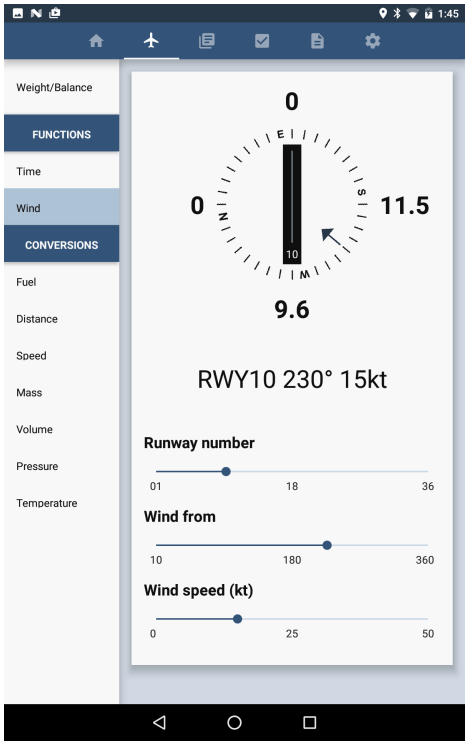

Figure 4.16: Wind speed calculator.

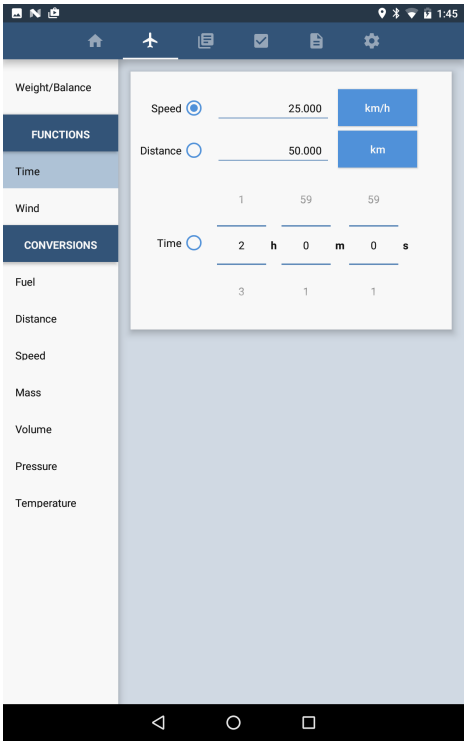

Figure 4.17: Flight time calculation.

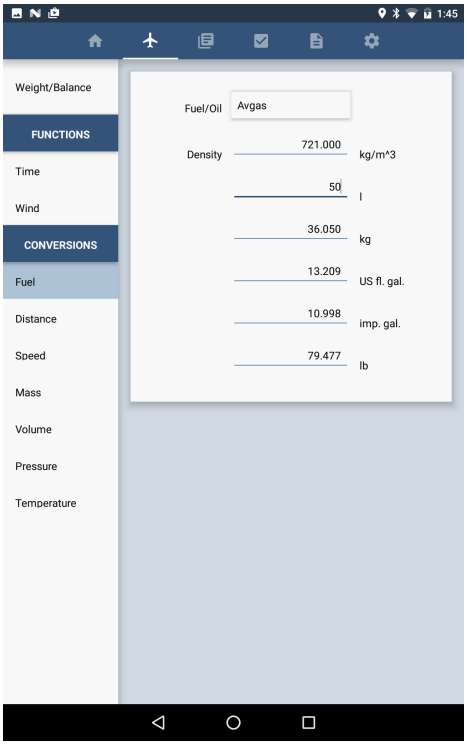

Figure 4.18: Fuel weight calculator.

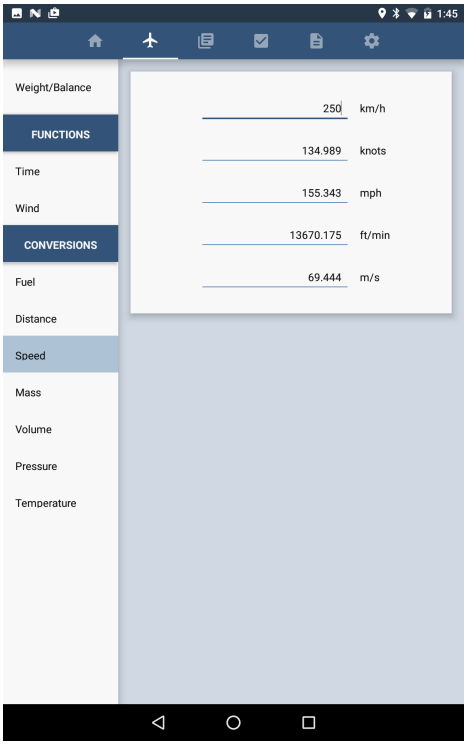

Figure 4.19: Unit conversion.

An other useful calculator is the *Time* function, that allows to calculate distance, speed and time (figure 4.17). The variable that is selected by a `RadioButton` will be calculated based on the other two entered values. It also allows to switch between different speed and distance units.

The `Fuel` conversion in this section allows to calculate the volume and weight of different fuel and oil types based on their density. This is visible on figure 4.18. The user is able to select from some preselected fuel types, or can enter a custom fuel/oil density that will be used for the calculation.

The other supported unit conversions are `Distance`, `Speed`, `Mass`, `Volume`, `Pressure`, and `Temperature`. Their layout is visible on figure 4.19.

All the values are updated real-time in every function and converter, so there is no need to push an additional button for the values to be calculated.

4.2 Application Structure

The here implemented application is built up from several Java classes. The applications Java packet is called `sk.lgstudio.easyflightbag`. The following list names all the components and building blocks of the application inside this packet, and provides a brief description of their functions.

- `MainActivity` - The Activity of the application.
- `calculations`
  - `Calculator` - Implements an universal unit conversion screen.

  - `CalculatorData` - Class holding static values for unit conversion.
  - `CalculatorFuelOil` - Screen for the fuel and ail weight and volume calculator.
  - `CalculatorTemperature` - Screen for the temperature conversions.
  - `CalculatorTime` - Screen for the time/speed/distance calculator.
  - `CalculatorWB` - Weight and balance calculator.
  - `CalculatorWind` - Wind calculator view.

- `dialogs`
  - `AirplaneEditorDialog` - Form to edit an airplanes details.
  - `ChecklistEditorDialog` - Dialog to edit checklists.
  - `DeleteDialog` - Simple delete confirmation dialog.
  - `FlightPlanEditorDialog` - Dialog for plotting a flight plan on the map.
  - `FlightPlanDialog` - Creating new flight plan.
  - `FlightPlanInfoDialog` - Shows the basic information about the current flight.
  - `OverlayDetailDialog` - Shows the details of the airspaces and airports at a given location.
  - `PdfViewerDialog` - PDF file reader.
  - `SelectorDialog` - Lists the Json files or the sub-folders of a given folder based on call arguments.
  - `SplashDialog` - A splash-screen is showed while the application starts and the `MainActivity` initializes the data.
  - `WbGraphDialog` - Weight and balance calculation results.

- `fragments`
  - `FragmentAIP` - Implements the AIP viewer screen.
  - `FragmentCalc` - Implements the the calculations screen and uses the components from the `calculations` packet.
  - `FragmentChklist` - Implements the checklist screen functionality.
  - `FragmentDocs` - Simple document browser screen.
  - `FragmentHome` - The home fragment that is used for navigation.
  - `FragmentSettings` - Application settings section

- `managers`
  - `AIPManager` - Class that manages the AIP documents and their downloading.
  - `AirplaneManager` - Holds the data abut the selected airplane, reads and saves Json files.
  - `FlightPlanManager` - Holds the data about a flight plan, reads and saves Json files.
  - `MapOverlayManager` - Holds the data of the airspaces and airports that are visible on the map

- `openAIP`
  - `Airport` - Airport `.aip` file parser.
  - `Airspace` - Airspace `.aip` file parser.
- `services`
  - `AIPDownloader` - Abstract class for downloading the AIP PDFs from a given source.
  - `AIPcz` - Downloads the Czech AIP from the ŘLP AIP service [1].
  - `BTTrackerService` - Service to get the actual position data from a *Bluetooth* source.
  - `GPSTrackerService` - Service that receives the actual position from the GPS.
  - `OpenAIPDownloader` - Downloads files from the `openAIP` database.

The core of the application int the `MainActivity`. It contains the tab switching buttons on the top and the `FrameLayout` where the selected content is loaded. These contents are implemented as separate `Fragments`. It is also responsible for starting and stopping the `GPSTrackerService` and the `BTTrackerService` based on the chosen option in the `Settings`. These services are sending `Intents` with the actual location with a one second interval. These `Intents` are received by the `MainActivity`, parsed, then sent to the `FragmentHome` for further work. The `MainActivity` also initializes the `AIPManager`, the `AirplaneManager` and the `MapOverlayManager` on the start of the application and pases them to the `Fragments` using them, so there is only one instance existing.

`FragmentHome` (visible on figure 4.8) is the `Fragment` containing the navigation map and showing the flight data. Its main components are the `MapView`, the `ListView` containing the route way-points, and the layout showing the speed, elevation and the bearing values. It also contains `Buttons` to open the `FlightPlanDialog`, the `FlightPlanInfoDialog` (figure 4.9) or to manipulate with the map.

Every here mentioned `Dialog`'s return values are handled by their `onDismiss` and `onCancel` listeners implemented by the calling class.

The `FlightPlanDialog` (on figure 4.6) is used to plan a new flight and returns a `FlightPlanManager` object if a plan was created. Based on this flight plan the `MapView` and the list for the way-points is filled. This `Dialog` has three section. One is used for specifying the airplane and it's properties, one for the flight plan and the last sums up some details about the chosen plan. Both the airplane and the flight plan selector button opens the `SelectorDialog`, but with different arguments. The `SelectorDialog` was design to create, delete, and list `.json` files or sub-folders of a folder, based on the constructors parameters. It returns the selected item as a `File` object. When a new item was created or the item was selected to be edited, an additional `boolean` is set. By this value the `AirplaneEditorDialog` (figure 4.3) manipulating with the `AirplaneManager` or the `FlighPlanEditorDialog` (figure 4.7) working with the `FlightPlanManager` is opened. Both the airplane and the flight plan is stored in a `.json` file. The example of their structure is visible in listings 4.1 and 4.2.

```
{
  "eq_type":"SportStar RTC",
  "eq_color":"White",
  "eq_com_nav":"DG",
  "eq_ssr":"CS",
  "cruise_sp":108,
  "climb_sp":108,
  "descent_sp":108,
  "climb_rate":1020,
  "desc_rate":1020,
  "fuel_density":0.721,
  "fuel_flow":15,
  "max_takeoff":600,
  "max_landing":600,
  "empty_weight":375.8,
  "empty_arm":268,
  "arm_sat":false,
  "additional_weight": [
    ["Fuel", 771, 120, 0],
    ["Pilot", 512, 0],
    ["Co-Pilot", 512, 0],
    ["Luggage", 1083, 25]],
  "limits": [
    [250, 368.5],
    [313.5, 368.5],
    [362.5, 393.5],
    [400, 450],
    [400, 505],
    [400, 600],
    [375, 600]
    [331.25, 518.5]]
}
```

Listing 4.1: Airplane file example.

```
[
  {
    "name":"LKTB",
    "lat":49.151389,
    "lon":16.693889,
    "editable":false
  },
  {
    "name":"1",
    "lat":49.109797187549866,
    "lon":17.029361464083195,
    "editable":true
  },
  {
    "name":"2",
    "lat":49.00107803648605,
    "lon":17.00547706335783,
    "editable":true
  },
  {
    "name":"3",
    "lat":48.98837152768484,
    "lon":16.50390263646841,
    "editable":true
  },
  {
    "name":"LKTB",
    "lat":49.151389,
    "lon":16.693889,
    "editable":false
  }
]
```

Listing 4.2: Flight plan file example.

The second tab of the application opens the `FragmentCalc`, visible in figures 4.4, 4.16, 4.17, 4.18, and 4.19. This `Fragment` is made from a sub-menu list visible on the left, and a `FrameLayout` on the right. Based on the selected sub-menu, the given `View` is added into the frame. The added `View` is controlled by a `Calculator` object, which implements an universal unit conversion screen visible on figure 4.19. Weight and balance, wind, time and fuel calculation screens are implemented in their own class extending the `Calculator` class, an overriding some of its methods. The `CalculatorWB` class in addition implement some unique additions. Here it is possible to open the `SelectorDialog` and the `AirplaneEditorDialog` the same way it is possible during flight planning. This feature is however possible only if there is no ongoing active flight plan. Otherwise the airplane defined by the flight plan is selected. The `View` of the calculator is filled dynamically based on the airplanes properties. When all data are filled correctly the result is opened on the `WbGraphDialog` visible in figure 4.5.

The `FragmentAip` lists the *AIP* documents based on the generated `data.txt` file generated by the `AIPDownloader Service`. This file structure is shown in listing 4.3. The f`Fragment` dynamically loads the the categories and files into a `ListView` for a given level. For a new level of the structure a new `ListView` is created on the right side of its par-

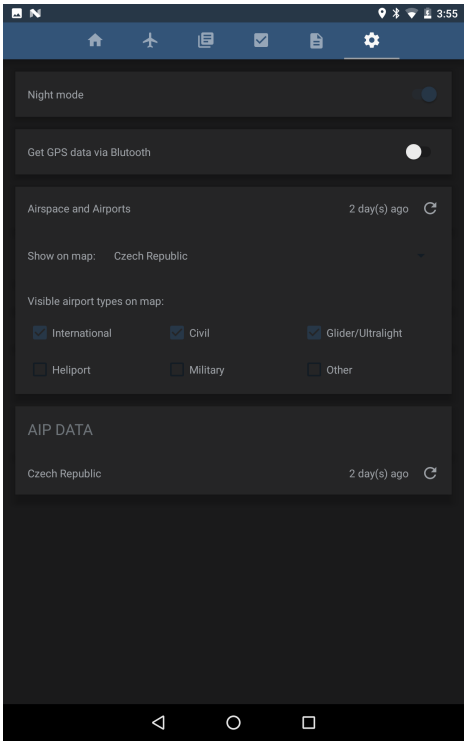

Figure 4.20: Settings window.

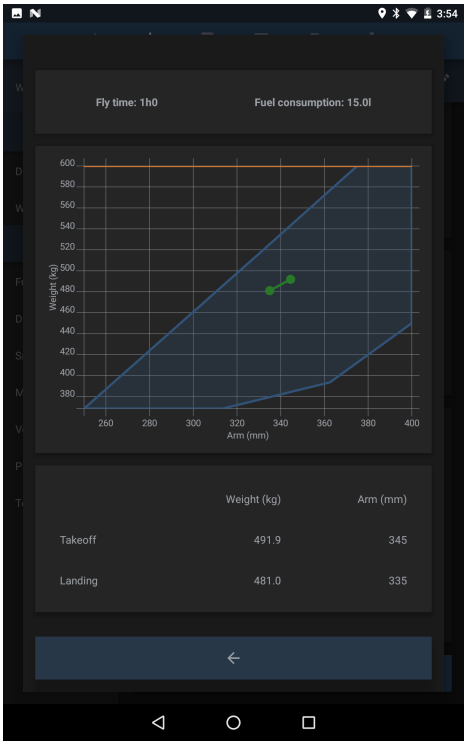

Figure 4.21: W&B in dark theme.

ent when an item was clicked in a list. The selected document items are opened by the `PdfViewerDialog`. All the opened `.pdf` documents last opened page position is saved to the applications `SharedPreferences`, so the user can continue the file browsing where he finished it the last time.

The `Fragment` opened of the fourth tab is the `FragmentChklist`. This `Fragment`'s `View` is divided into two columns. The left is a `ListView` listing `.txt` files in a given folder. This list has a header, where it is possible to open the checklists for different airplanes by the `SelectorDialog`. Every airplane has its own folder in the `/EasyFlightBag/Checklists/` folder. Here the checklists are saved as individual a `.txt` files. Every single line of the text file is considered as a new task in the list. Then a file was selected in the list on the left, the list gets divided into three parts. The completed tasks, the ongoing task, and the following tasks (figure 4.13). The green floating `ImageButton` on the right bottom of the window is used to check the task in the list. It is possible to add or edit the actual list by clicking on the add or edit button in the left list header. This action opens the `ChecklistEditorDialog`.

The `FragmentDocs` implements a simple `View` containing a `ListView`. This lists the `.pdf` files found in the `/EasyFlightBag/Documents/` folder. The selected item is opened by the `PdfViewerDialog`.

The `FragmentSettings` allows to configure, and update the offline data used by the application (figure 4.20). This `Fragment` contains four sections. The first is toggling between the day and night layout colors. Examples of the dark color theme are shown in figures 4.20 and 4.21. This setting is saved to the applications `SharedPreferences`, so then the application is restarted, the last used theme will be preselected. The second is used to turn on the *GPS* simulation via *Bluetooth*. This feature's details are described in section 4.3. The

following card is responsible for the map overlays such as airspaces and airports. These data are downloaded form the *openAIP* database [20]. Refreshing the database is handled by the `MapOverlayManager` that starts the downloader service and creates a notification about the ongoing download. The downloaded `.aip` files are stored in the `/EasyFlightBag/Airspace` folder and parsed by the `Airspace` and `Airport` objects based on the specification published on the *openAIP* website ([21], [22]). Here the user can also filter the visible airport types and select a specific country to be shown on the map. Selecting a single country makes the map layout in planning (section 4.1.2) and navigation (section 4.1.3) faster and more responsive. The last function of the setting is for the *AIP* document update. The only *AIP* to be updated is from the official AIP provider for the Czech Republic [1]. The download is started by the `AIPManager` that also creates a notification about the ongoing download. The downloaded documents are listed into a `data.txt` document by the `AIPDownloader Service`, and used by the `FragmentAIP` (section 4.1.5) to visualize the *AIP* structure. The structure of this file is visible in listing 4.3. Every line contains a pair or a triplet of values divided by a „:“ character. The first number represents the level of the node and the second the name of the node. In case the node is a file, the third value contains the name of the `.pdf` file.

```
0:"First main category"
1:"First file in category":1a.pdf
1:"Second file in category":1b.pdf
...
1:"First subcategory"
2:"First file in subcategory":11a.pdf
2:"Second file in subcategory":11b.pdf
...
1:"Second subcategory"
2:"First file in subcategory":12a.pdf
2:"Second file in subcategory":12b.pdf
...
0:"Second main category"
...
```

Listing 4.3: AIP data file structure.

4.3 GPS simulation via Bluetooth

The application is possible to use indoors for testing purposes in flight simulators. By turning on this mode in the `Settings` the `BTTrackerService` is started. It creates a *Bluetooth* server that is waiting for a device to connect. The server is receiving a string with the value sequence shown in listing 4.4 divided by a ' ' character.

```
[longitude] [latitude] [altitude] [course] [ground speed]
```

Listing 4.4: *Bluetooth* input string data sequence.

For this purpose a `Python` script example was implemented, that receives the data from *TCP* server named *AW-COM*[16] and sends a string of values via *Bluetooth* to our

application. This kind of communication was used in the motion simulator based testing further described in chapter 5.2.2.

4.4 Third-party Libraries and Dependencies

The implemented application uses the following libraries.

HelloCharts

The *HelloCharts*[14] is a highly customizable chart drawing library featuring many different types of charts. In this project it is used by `WbGraphDialog` where it is used to visualize the airplanes longitudinally center of gravity inside the center of gravity limits box.

Android PDF Viewer

The *Android PDF Viewer*[7] is used to open and render /textitPDF files. It is used by the `PdfViewerDialog` to show the selected *AIP* or flight manuals. It enables fast scrolling through pages and zooming using the double-finger zoom gesture.

Jsoup

The *Jsoup*[15] library is a powerful parser for *DOM* (*Document Object Model*) based documents like *HTML* and *XML*. It is used here to get and parse the *AIP* website provided by *ŘLP*[1] in the `AIPcz Service`. It is also used to parse the airport and airspace `.aip` files provided by the *openAIP* database [20] in the `Airplane` and `Airspace` classes.

GoogleMaps

The *Maps API*[17] is an important part of the *EasyFlightBag* application. All the map bases are provided by this API. It is used in the `FragmentHome` and `FlightPlanEditorDialog`.

This API allows the application to show and interact with the map. It also handles and draws the used map layouts like the airspace polygons, airport icons and flight plan lines.

The displayed map is in *Web Mercator* projection that was standardized by the *National Geospatial-Intelligence Agency*[29].

Due to this API the application relies on the system's built-in *Google*'s *Maps* applications cache and on the internet connection to load the map. This internet connection dependency can be fixed by saving the needed map areas for offline use in the *Maps* app's *Offline areas* menu.

Chapter 5

Testing and Evaluation

EasyFlightBag was tested both in laboratory and in real life situations and evaluated based on different criteria. This chapter sums up the conditions and the results of these tests starting with some basic application requirements.

5.1 Hardware and Software Requirements

The application was designed to run on *Android* devices, mainly tablets running at least version 5 (API 21). The device size is not limited, but it is recommended to use at least one with a 7" display with a *1920x1080 pixel* resolution. The device also has to be equipped with an internal *GPS* module, internet connection and *Bluetooth* for the *GPS* simulation mode.

Based on *Android*'s resource manager the *EasyFlightBag* application uses around *100 MB* of *RAM* while running in a navigation mode. The maximal *RAM* usage occurs while updating the *AIP* database. In this case the application can use around *160 MB* of *RAM*.

Based on the testing done it is not possible to determine the minimal *CPU* requirements, but it was running solid on a *Dual-Core 2.2GHz ARM* chip without any noticeable lag. The most *CPU* consuming task of the airport icon generation may however slow down the application response time right after a new airport data were downloaded. This is however only one-time inconvenience during the runtime.

A free storage space of at least *100 MB* is also recommended. This is used for storing the Czech *AIP* documents of about *85 MB*, downloaded airspace and airport files, and saved airplane and flight plans. In case of adding own documents into the `/EasyFlightBag/Documents/` folder, the application occupies more storage space.

An additional not obligatory addition are the saved offline maps provided by the *Google Maps* application. These maps also require som storage space. A *100 km* x *100 km* area takes up an approximately *75 MB*. In case no offline maps are saved, the application further needs a continuous internet connection during flight so the base map can be downloaded on-demand.

5.2 Testing

EasyFlightBag's functions were tested during the development using a *Nvidia Shield K1* tablet device with the following specification:

- CPU: Quad-core 2.2 GHz Cortex-A15.
- GPU: ULP GeForce Kepler.
- Chipset: Nvidia Tegra K1.
- RAM: 2 GB.
- Internal Storage: 16 GB.
- OS: Android 7.0 (Nougat).
- Display: 8 inch, 1920x1200 ISP LCD capacitive multi-touchscreen.
- Hardware features: A-GPS, GLONASS, WiFi a/b/g/n, Accelerometer, Gyro.

5.2.1 Laboratory Testing

All the applications features were tested during the development so they work seamlessly and correctly.

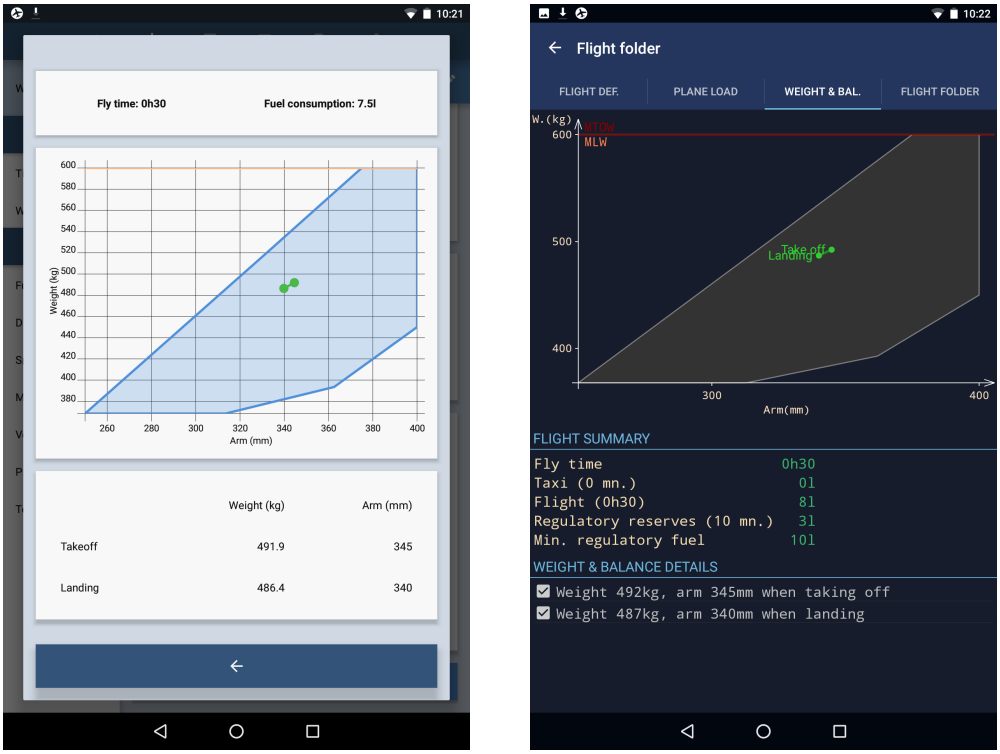

Figure 5.1: *EasyFlightBag* (left) W&B test result compared to *Flight Assistant*'s (right).

The W&B calculations results were compared with other application with this feature (mentioned in section 2.2) so the values match. Figure 5.1 shows the results compared

to the *Flight Assistant* application's. In both cases the same values were added to the calculator. These values are the same tat we can find in listing 4.2. On the result graphs and in the details we can see that the results differ only due to the rounding error.

For other calculations and unit conversion validation there were numerous online converters used to make sure the application gives in every situation the right results.

The *GPS* precision and the map orientation, speed and heading were tested by car on the ground to make sure the right values are displayed and the map's automatic rotation and follow function works precisely.

5.2.2 Simulator Based Testing

The second phase of the test was performed on a flight simulator located at the *Faculty of Information Technology* of *Brno University of Technology*. This simulator made it possible to test the application in simulated flight with experienced pilots.

The *Android* device was connected to the simulator via the *AW-COM*[16] server that provides the requested data for the *Bluetooth* client application (described in section 4.3) as it is visible on figure 5.2.

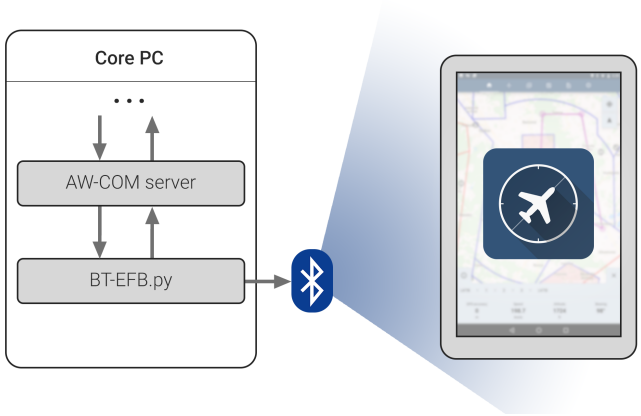

Figure 5.2: Flight simulator communication diagram.

In this test phase the application was tested for stability during a simulated flight. The *EFB* was used in an airplane cockpit and as tested for the information visibility, readability, and display precision.

5.3 Evaluation

The application was evaluated from three points of view. The first point investigates if the application fulfills the predefined feature requirements. This is listed in section 5.3.1. Then we test if the user interface is as usable and intuitive as it was planned to be. For this purpose a group a people was asked to complete a list of tasks. This evaluation process is described in section 5.3.2. The third evaluation (section 5.3.3) was created based on an official evaluation process so see if the application fulfills the requirements imposed on an *Electronic Flight Bag*.

5.3.1 Evaluation of the Features

The application design section 3.2, includes the definition of features. Based on these we, were able to evaluate if the application complies to these defined requirements.

The following functions fulfill the predefined requirements.

- The possibility to define and save different airplane profiles.
- W&B calculation for airplanes.
- Unit conversions.
- Checklist creation and management.
- *AIP* browsing capability.
- Airplane manual browsing.
- Works in most Central European countries.
- Providing airspace and airport information.
- Flight planing.
- Navigation assistance.

The features which were not implemented are the following.

- Providing weather information (*METAR*).
- Actual *NOTAM*, *AUP*, and *TAF* browsing.

These functions did not get into the application due to the lack of a free to access database that would provide actual information for the required countries.

5.3.2 Evaluation of the User Interface and Usability

This evaluation was crated based on the ISO/IEC 9126-4 [19] effectiveness metrics.

The user interface usability metrics were created based on 20 user test where 10 of the participants were pilots (of general or commercial aviation). The participants were given a brief introduction to the application without showing them the actual user interface so an actual first reaction could be evaluated. The participants were given five tasks to complete.

1. How long is the main concrete runway of the Brno Tuřany Airport (LKTB).
2. Create a new checklist for a new airplane with three tasks, then provide them.
3. Create a new airplane profile based on the given data.
4. Create a flight plan from Brno Tuřany (LKTB) to Bratislava (LZIB).
5. How much weights 50 liters of *Avgas* fuel.

The participants were observed during the completion of the tasks and the time of completing the tasks was also measured. These time measurements however are not authoritative in some cases where the participants were asked to type in data from a printed paper, as the typing speed of the the involved participants highly differed. Therefore, these times were not counted, only the success rate considered.

Effectiveness and Efficiency

Based on the ISO/IEC 9126-4 metric the application scored a 100% completion rate as every 20 participants completed all the tasks

The error rate was difficult to define in this situation, as it is impossible to define what counts as an attempt. As the participants were not given any introduction about the applications layout, they had to find the right place in the application to complete the task. Therefore the error rate is included into the time efficiency of the application. The efficiency would be however much higher with experienced users.

1. As the home-screen contains a map view, the participants were automatically browsing the map and looking for the given airport. It took on average only about 15 seconds to find the requested information for a first time user.

2. The second task required to find the location of the checklists. This made the participants search for the checklists on an average of 20 seconds. After the right section was found, it took on average about a 1 minute and 10 seconds to fully finish this tasks.

3. From the checklists is usually took over 15 seconds until the user found a place to add a new airplane profile. From here it usually took two minutes until the user entered all the data from a printed form. This time however is not very relevant as these data are filled only once and their quick access is not requested during the flight.

4. The task to create a flight plan took on average of 1 minute to complete. Many off the pilots were however confused when the application requested to add a name for the new flight plan.

5. Before the completion of the last task it took on average 15 seconds to find the right converter for this purpose. Then it only took on average of 7 seconds to get the right answer.

Overall the efficiency is considered to be good even with first time users. The implemented layout and icons are clear and the users easily understand them.

Satisfaction

The user satisfaction was evaluated based on the marks from 1 (good) to 10 (bad) given by the users for different aspects. The result of the user ratings is visible in 5.1. The pilots in the control group were given additional questions about he applications features. These questions and their results are showed in table 5.2.

Aspect	1	2	3	4	5	6	7	8	9	10
Interface design	30%	45%	20%	5%	0%	0%	0%	0%	0%	0%
Ease of use	15%	50%	25%	10%	0%	0%	0%	0%	0%	0%

Table 5.1: User interface satisfaction of the control group.

Based on the marks and the answers of the participants the application is considered to be a visually well designed application. The participants did not found it hard to navigate in the application even during the first time use.

Some of the pilots were missing some features that were also mentioned in section 5.3.1, but overall they found the application nicely made to be used for recreational aviation.

Question	Yes	Maybe	No
Do you like the features of the application?	80%	20%	0%
Would you consider using this application as your *EFB*?	70%	30%	0%

Table 5.2: Feature satisfaction of the asked pilots.

5.3.3 Evaluation of the Electronic Flight Bag Requirements

The application as an *EFB* system was evaluated based on the „*Best practices for evaluation and use of EFB*“[13] research released by the *EASA*. As this document was designed generally for airplanes of all sizes, many of it's requirements goes over the limits of *GA* and of the lightweight airplane operation.

Based on this document the two most important aspects to evaluate an *EFB* are connected to the flight performance, and to the list of hazards of an *EFB* system.

Airplane Performances

There is a number of conditions that have to be accounted while calculating the take-off or landing performance of an airplane. Some of these conditions are listed below:

- Weight.
- Flaps settings.
- Engine power and settings.
- Runway condition and slope.
- Wind speed and direction.
- Outside temperature and pressure.
- Obstacles and abnormal conditions.

The application's W&B calculations does not account to all of these aspects so it does not fulfill the recommended calculation precision defined by the *EASA*. This means that the application is not suited for performance calculations required for medium or heavy airplanes. On the other hand *EasyFlightBag* is aimed for the lightweight airplane pilots in *GA* where it's capabilities are satisfactory.

List of Hazards

The following list sums up the hazardous aspects of an *EFB* system, and describes the implemented solution to prevent the problem occurrence.

- Wrong aircraft selected: The flight crew is responsible to select the right airplane before the flight.
- Error in digitalizing the paper flight manual: The flight crew is responsible to carefully enter the right data from the original flight manual.

- Airport database out-of-date: The last update time of the database is indicated in the application's settings.
- Errors in manual real-time changes: The application does not provide recalculations for real-time changes during the flight.
- Perception that airport manual is up-to-date while it is not: The update time is saved only when the update succeeded.
- Airport database error: No airport data are displayed on the map.
- Wrong database stored: The user is unable to selectively update the database.
- Database storing errors at bit-level: Handled by the *Android OS*.
- Introducing viruses: Handled by the *Android OS*.
- Corruption of databases: No data is displayed when the database is corrupted.
- Database updating during flight: No automatic update feature is implemented.
- Inaccuracy of the data: The accuracy of the airport and airspace details depends on the *openAIP* [20] database.
- Wrong output due to software settings: No settings affect calculation output values.
- Wrong output due to user interference: The application allows only single user use.
- Wrong output doe to unit conversion errors: All the units are indicated in every situation.
- Wrong output due to interference with other applications: The application does not rely on any other application so no other software interfere with any.
- Data from previous flight used: The application does not store previous flight data.
- Error in unit conversions: All unit conversions were properly tested so the results are right.

Although the application was implemented to solve most of the hazardous aspects, it is not enough to introduce a new *EFB* system. Before the use, the operator also has to evaluate if the given system is suitable for the needs. An example of this kind of evaluation is showed in appendix A. This evaluation checklist was created based on an advisory publication of the Australian Government Civil Aviation Safety Authority[12].

Chapter 6

Future Work and Conclusion

While *EasyFlightBag* fulfills the requirements and the defined goals, there is always a room for improvement and for future development. In the following section there are described some ideas which would make positive addition to the application.

6.1 Future Work

For the future the most important would be to add the missing meteorological information in addition with the *NOTAM*, *AUP*, and *TAF* browsers so the pilot gets access to more important information during the flight. As these information are connected to the individual airports or airspaces it would be beneficial to include them amongst their already showed details (Described in section 4.1.3).

Another field to improve is related to the *AIP* browser, where the inclusion of other countries *AIP*s would be beneficial for better usability outside of Czech Republic.

An other feature for improvement could be the implementation of a inclusion terrain maps and three-dimensional maps to assist flights in mountainous areas. This could help pilots in obstacle avoidance at lower altitudes and in lower visibility conditions.

There are many other functions that would improve the actual application but the most important is that no feature should distract or mislead the pilot during the actual piloting.

6.2 Conclusion

In conclusion we can say the application successfully implements an easy to use and useful *EFB* for sport and recreational pilots in Central Europe. It helps in preflight preparations and provides important information and intuitive navigation interface during the flight.

The logical work-flow is the biggest benefit of the application starting with selecting the airplane, and plotting the route on the map, and then calculating the airplanes weight and balance distribution based on the previously set values.

Other benefits of the *EasyFlightBag* is that it was implemented using the *Android Material Design* [18] guidelines, so for every *Android* user the interface is very familiar and intuitive. The clean design of the navigation assistance is also considered as a positive feature so the pilot can see the important data with no distraction by an overfilled screen.

Overall the application is considered to be a useful tool that is worth to continue developing and to be released for public use.

Bibliography

[1] Aeronautical Information Publication. Řízení letového provozu ČR, s.p. [online]. http://lis.rlp.cz/ais_data/www_main_control/frm_en_aip.htm. cit [2017-01-02].

[2] Android. [online]. https://www.android.com [cit. 2017-01-02].

[3] Android API Guides. [online]. https://developer.android.com/guide/index.html. [cit. 2017-01-02].

[4] Android Dashboards. [online]. https://developer.android.com/about/dashboards/index.html. [cit. 2017-01-02].

[5] Android Developpers site. [online]. https://developer.android.com/index.html. [cit. 2017-01-02].

[6] Android (operaiting system). [online]. https://en.wikipedia.org/wiki/Android_(operating_system). [cit. 2017-01-02].

[7] Android PDF Viewer. [online]. https://github.com/barteksc/AndroidPdfViewer.

[8] Android Platform Architecture. [online]. https://developer.android.com/guide/platform/index.html. [cit. 2017-01-02].

[9] Android Play Store. [online]. https://play.google.com/store/apps. cit [2017-01-02].

[10] Android version history. [online]. https://en.wikipedia.org/wiki/Android_version_history. [cit. 2017-01-02].

[11] Apple App Store. [online]. https://itunes.apple.com/us/genre/ios/id36?mt=8. cit [2017-05-02].

[12] Australian Goverment Civil Aviation Safety Authority. Electronoc Flight Bags. 2013. [online]. https://www.casa.gov.au/file/104846/download?token=jf6XFfT6. cit [2017-05-02].

[13] EASA. Final Report EASA_REP_RESEA_2014_1. Best practices for evaluation and use of EFB. 2014. [online]. https://www.easa.europa.eu/system/files/dfu/EASA-Research-Rep-2014-1.pdf. cit [2017-05-02].

[14] HelloCharts. [online]. https://github.com/lecho/hellocharts-android.

[15] Jsoup. [online]. https://jsoup.org/.

[16] Karol Rydlo, Vizualizace Vzdušného Prostoru ve 3D, Diplomová práce, Brno, MUNI-FI v Brně 2012. https://is.muni.cz/th/389970/fi_m/diplomova_prace_389970.pdf. cit [2017-05-02].

[17] Maps Android API. [online]. https://developers.google.com/maps/documentation/android-api/. [cit. 2017-05-02].

[18] Material design. [online]. https://material.io/guidelines/. cit [2017-01-02].

[19] Nigel Bevan. ISO and Industry Standards for User Centred Design. 2000. [online]. http://usabilitynet.org/trump/documents/Usability_standards.ppt.pdf. cit [2017-05-02].

[20] OpenAIP. [online]. https://www.openaip.net. cit [2017-05-02].

[21] OpenAIP, Airport. [online]. https://www.openaip.net/system/files/openAIP_aip_format_1_1_airport.pdf. cit [2017-05-02].

[22] OpenAIP, Airspace. [online]. https://www.openaip.net/system/files/Airspace.pdf. cit [2017-05-02].

[23] RTCA: DO-178B Software Considerations in Airborne Systems and Equipment Certification. [online]. 1992-12-01. http://www.rtca.org/store_product.asp?prodid=581. [cit. 2017-01-02].

[24] U.S. Department of Transportation Federal Aviation Administration: AC 120-64 - Operational Use and Modification of Electronic Checklists. [online]. 1996-04-24. https://www.faa.gov/documentLibrary/media/Advisory_Circular/ac120-64.pdf. [cit. 2017-01-02].

[25] U.S. Department of Transportation Federal Aviation Administration: AC 91.21-1B - Use of Portable Electronic Devices Aboard Aircraft. [online]. 2006-08-25. https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_91.21-1B.pdf. [cit. 2017-01-02].

[26] U.S. Department of Transportation Federal Aviation Administration: AC 91-78 - Use of Class1 or Class 2 Electronic Flight Bag (EFB). [online]. 2007-07-20. https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_91_78.pdf. [cit. 2017-01-02].

[27] U.S. Department of Transportation Federal Aviation Administration: AC 20-173 - Installation of Electronic Flight Bag Components. [online]. 2011-09-27. https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC%2020-173.pdf. [cit. 2017-01-02].

[28] European Aviation Safety Agency: AMC 20-25 -Airworthiness and operational consideration for Electronic Flight Bags (EFBs). [online]. 2014-02-09. https://www.easa.europa.eu/system/files/dfu/2014-001-R-Annex%20II%20-%20AMC%2020-25.pdf. [cit. 2017-01-02].

[29] National Geospatial-Intelligence Organization. Standardization document: Implemetation pactice - Web Mercator Map Projection. [online]. 2014-02-18. http://earth-info.nga.mil/GandG/wgs84/web_mercator/%28U%29%20NGA_SIG_0011_1.0.0_WEBMERC.pdf. cit [2017-05-02].

[30] U.S. Department of Transportation Federal Aviation Administration: AC 120-76C - Guidelines for the Certification, Airworthiness, and Operational Use of Electronic Flight Bags. [online]. 2014-05-09. https://www.faa.gov/documentlibrary/media/advisory_circular/ac_120-76c.pdf. [cit. 2017-01-02].

[31] EASA: European Aviation Safety Agency. [online]. 2017. https://www.easa.europa.eu. [cit. 2017-01-02].

[32] FAA: Federal Aviation Administration. [online]. 2017. http://www.faa.gov. [cit. 2017-01-02].

Appendix A

Self Evaluation Checklist for the Introduction of *EFB*

Hardware

Do the physical characteristics of the proposed device make it suitable for use as an *EFB*?	**Yes**
Will the display be readable in all the ambient lighting conditions, both day and night, encountered on the flight deck?	**Yes**
Testing has been conducted to confirm EMI/EMC compatibility.	**Yes**
Is the format of the *EFB* suitable for the intended application?	**Yes**
Has the *EFB* been tested to confirm operation in the anticipated environmental conditions?	**Yes**
Does charging cause the *EFB* to heat above ambient temperature?	**No**
During operations in warm climates, can the operating temperature of the *EFB*, whilst charging, rise above OEM specifications/recommendations?	**No**
Has the internal battery of the *EFB* sufficient capacity to function for the maximum duration of operations anticipated?	**Yes**
Does the *EFB* require any external connectivity to function?	**No**

Software

Does the software application/s installed on the *EFB* enable it to replace documents and charts required to be carried on board the aircraft?	**Yes**
Does the software application/s proposed require regulatory approval prior to operational use?	**No**
Has the software application been evaluated to confirm that the information being provided to the pilot is a true and accurate representation of the documents or charts being replaced?	**Yes**
Has the software application been evaluated to confirm that the computational solution/s being provided to the pilot is a true and accurate solution?	**Yes**

Are there other software applications intended to support any additional requirements of the operator?	**No**
Does the software application/s have adequate security measures to prevent unauthorised database modifications and prevention of contamination by external viruses?	**Yes**

Installation

If *EFB* is hand held, can it be easily stowed securely?	**Yes**
When stowed, is the *EFB* readily accessible in flight?	**Yes**
Is the mounting device compliant with the applicable crashworthiness requirements?	**Yes**
Has the installation of the mounting device been approved in accordance with the appropriate airworthiness regulations?	**Yes**
If the mounting device for the *EFB* is moveable, can it be easily be locked in place?	**Yes**
Has provision been provided to secure or lock the mounting device in a position out of the way of flight crew operations?	**Yes**
Is there any evidence that there is mechanical interference issues with the mounting device, either on the side panel (side stick controller) or on the control yoke in terms of full and free movement under all operating conditions and non-interference with buckles etc?	**No**
If *EFB* mounting is on the control yoke, have flight control system dynamics been affected?	**No**
For fixed mounts, has it been confirmed that the location of the mounted *EFB* does not obstruct visual or physical access to aircraft displays or controls or external vision?	**Yes**
For fixed mounts, has it been confirmed that the mounted *EFB* location does not impede crew ingress, egress and emergency egress path?	**Yes**
Does the mounted *EFB* allow easy access to the *EFB* controls & *EFB* display?	**Yes**
Does a dedicated power outlet for powering/charging the *EFB* need to be fitted?	**Yes**
Is there a means other than a circuit breaker to turn off the power outlet?	**No**
If the *EFB* has an alternate backup power source, does the backup source have an equivalent level of safety to the primary power source?	**No**
Have guidance/procedures been provided for battery failure or malfunction?	**No**
Does the *EFB* cabling present a hazard?	**No**
Is there a means to secure any cabling?	**Yes**
Is stowage readily accessible in flight?	**Yes**
Does the stowage cause any hazard during aircraft operations?	**No**

Usability

Is the *EFB* mount easily adjustable by flight crew to compensate for glare and reflections?	**Yes**
Can the brightness or contrast of the *EFB* display be easily adjusted by the flight crew for various lighting conditions?	**Yes**
Is the hand held *EFB* easily stowed in an approved receptacle during flight?	**Yes**
Is there an easy means for the flight crew to turn off the *EFB* in the event of a failure?	**Yes**
Does the location of the *EFB* interfere with any normal or emergency procedures?	**No**
Does the protective screen (if fitted) interfere with the viewing of the *EFB* or the ability to manipulate the cursor?	**No**
Is there an easy way to recover the configuration of the *EFB* back to the default settings, as controlled by the EFB administrator, in the event of a failure?	**Yes**
Can the flight crew easily determine the validity and currency of the software installed on the *EFB*?	**Yes**
When hosting a variety of applications on the *EFB* is the flight crew able to make a clear distinction between flight and non-flight related activities?	**Yes**

Administration

Is the person nominated to administer the *EFB* suitably trained?	**Yes**
Do the listed responsibilities match the requirements of the system?	**Yes**
Are there adequate resources assigned for *EFB* administration?	**No**
Are there appropriate procedures for all phases of flight?	**Yes**
Are the procedures clearly presented, suitably illustrated and readily understood?	**Yes**
Is there a clear description of the system, its operational philosophy and operational limitations?	**Yes**
Have crew procedures for *EFB* operation been integrated with existing *Ops* manual?	**No**
Are there suitable crew cross-checks for verifying safety-critical data?	**Yes**
Is there any additional workload mitigated/controlled? Do crew procedures include a requirement to verify the revision status of software and data?	**Yes** **Yes**
Do the procedures cover system re-boots, lock-ups and recovery from incorrect crew actions?	**Yes**
Are there procedures/guidance for loss of data and identification of corrupt/erroneous outputs?	**Yes**
Are there contingency procedures for total or partial *EFB* failure?	**Yes**

Is the procedure in the event of a total *EFB* failure available outside the *EFB*	**Yes**
Have the *EFB* redundancy requirements been incorporated into the *Ops* Manual?	**Yes**
Are flight crew members and ground staff training programs fully documented?	**Yes**
Is the training methodology matched to the participant's level of experience and knowledge?	**Yes**
Has the operator assigned adequate resources (time/personnel/facilities) for training in operation of *EFB*?	**Yes**
Is there access to actual or simulated *EFB* equipment for interactive training?	**Yes**
Does the training material match the *EFB* equipment status and published procedures?	**Yes**
Does the training program include human factors in relation to *EFB* use?	**Yes**
Does the training program incorporate training system changes and upgrades in relation to *EFB* operation?	**Yes**
Does the training material match the *EFB* equipment status and published procedures?	**Yes**
Are there controlled, documented procedures for the control of hardware and component stocks?	**Yes**
Do the procedures include repair, replacement and maintenance of *EFB* equipment and peripherals?	**Yes**
Do the procedures include validation following repair?	**Yes**
Are there documented procedures for the configuration control of installed software?	**Yes**
Are the access rights for personnel to install or modify software components clearly defined?	**Yes**
Are there adequate controls to prevent user corruption of operating systems and software?	**No**
Are there adequate security measures to prevent system degradation, viruses and unauthorised access?	**No**
Are procedures defined to track database expiration and install chart database updates?	**No**
Are there documented procedures for the control and management of data?	**Yes**
Do the procedures interface with procedures used by external data providers?	**No**
Are the access rights for users and administrators to manage data clearly defined?	**Yes**
Are there adequate controls to prevent user corruption of data?	**Yes**
Does the operator allow private use of the *EFB*?	**Yes**
Does the operator have a policy on private use?	**No**

Appendix B

Content of the CD

The attached CD contains the following file structure.

- `/EasyFlightBag.apk` - application install executable
- `/Electronic_Flight_Bag.pdf` - documentation
- `/src` - source files
  - `/app` - application source files
  - `/latex` - documentation source files
  - `/bt-client` - Python source for *Bluetooth* connection
- `/data` - test data for the application
  - `/EasyFlightBag` - folder to be copied into the devices internal storage