ASSIGNMENT OF MASTER’S THESIS

Title:	Cross-platform mobile application for safer drone operations
Student:	Bc. Jan Matějka
Supervisor:	Ing. Lukáš Brchl
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of winter semester 2021/22

Instructions

The thesis aims to analyze, design, and implement a mobile application for safer drone operation. Its primary function is to provide the user with information about flight zones and other drones. The application will also include user registration and login functionality, adding drones, flight planning, and flight history. The application should be implemented in the Flutter framework for Android and iOS and will integrate with the existing backend of Dronetag s.r.o. company.

- Do research about existing solutions for safe drone operation
- Study the existing Dronetag backend interface
- Analyze, design and describe the application architecture
- Use the existing Dronetag backend to implement the mobile app
- Implement a sufficient number of tests to cover key application functionalities
- Document the code
- Test the application from a user perspective
- Evaluate the resulting application and propose its future extension

References

Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 17, 2020

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master’s thesis

Cross-platform mobile application for safer drone operations

Bc. Jan Matějka

Department of Software Engineering
Supervisor: Ing. Lukáš Brchl

May 28, 2020

Acknowledgements

I would like to thank my family, my girlfriend and friends for support while writing this thesis and all my studies. I also would like to thank my supervisor Ing. Lukáš Brchl, for this chance to realize me in mobile application development.

Declaration

In Prague on May 28, 2020 …………………

Citation of this thesis

Matějka, Jan. *Cross-platform mobile application for safer drone operations.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2020.

Abstrakt

Tato diplomová práce se zabývá vývojem multiplatformní mobilní aplikace ve frameworku Flutter pro bezpečnější provoz dronů. Práce obsahuje kompletní analýzu, návrh a samotnou implementaci aplikace, jejíž hlavní funkcionalitou je správa dronů a identifikačních zařízení připevněných k dronům, zobrazování zakázaných zón pro lety a plánování a sledování letů dronů.

V analýze je kladen důraz na rozbor existující webové platformy a celé infrastruktury, která poskytuje data klientským aplikacím a aplikacím třetích stran. Návrh obsahuje popis struktury aplikace a použité architektonické vzory frameworku Flutter. Implementace popisuje detaily o realizaci a obsahuje podrobný návod pro spuštění ve vývojovém, testovacím a produkčním prostředí. Zároveň obsahuje popis důležitých konfiguračních souborů, které se liší napříč různými prostředími a mají bezpečnostní charakter.

Klíčová slova Multiplatformní, mobilní, aplikace, dron, řízení, Flutter

Abstract

This master's thesis focuses on cross-platform mobile application development in the Flutter framework for safer drone operations. This thesis contains a full application analysis, design, and implementation. Its main functionality is the managing drones and identification devices attached to drones, showing restricted flight zones, and planning and watching drone flights.

The analysis emphasizes the analysis of the existing web platform and all infrastructure, which provides data to the client and third-party applications. The design contains a description of the application structure and uses architecture patterns of the Flutter framework. The implementation describes details about realization and contains detailed launch instructions in the development, test, and production environments. In addition, it contains a description of the necessary configuration files. These files differ across the various environments and have a security disposition.

Keywords Cross-platform, mobile, application, drone, operation, Flutter

Contents

List of Figures

Introduction

In 2012, drones began to spread around the world. At that time, they were mainly toys built by hobby pilots in the home. Thus, there were technical problems with them.

After a while, large companies began to see the commercial potential of drones and started to incorporate them into their visions for the future. Amazon introduced its autonomous drone concept to transport goods from logistics warehouses directly to customers. [3] As customer satisfaction increases even with small improvements to delivery time, Amazon has decided to produce its drone products, including the Prime Air delivery drone.

To make the delivery process more convenient and minimize costs, Amazon is developing an autonomous drone application. Hence, a significant increase in the number of autonomous aircrafts can be expected. Potentially, drones could be used to deliver medical materials in case of a major accident or disasters such as a hurricane, military conflict, or even during a disease epidemic like the current novel coronavirus pandemic.

With the increasing number of drone owners, problems have started to arise. Many hobby pilots have broken the law and endangered aviation safety when they were flying. Ignorance of laws was common because there had been no need for drone pilots to know any of the rules of aviation that traditional pilots must know. However, this problem has begun to be resolved. Some laws for traditional pilots are applicable to the drone pilot. Drone pilots need to know certain symbols and respect restricted areas, but they do not have to complete a full course of instruction like traditional pilots. For example, it is necessary for drone pilots to know the primary types and descriptions of zones for safe operation. These zones can be separated into different types based on their properties.

There are a variety of key parameters in the zones, for example, lower and upper altitude level. These flight levels are further divided into groups based on the altitude in feet. Furthermore, it is important to realize that the given altitude can be one of two types: AMSL (Above Mean Sea Level) and

AGL (Above Ground Level). This means it is necessary to be clear about the type of given altitudes, and in the case of AMSL, to be aware of the current altitude to avoid restricted areas.

Overall, airspace is divided into 7 classes, A–G. [4] There is an obligation to announce a flight from through class A–E to the authorities, and the Air Navigation Service of the Czech Republic approves the flight. The F and G classes are lower flight levels, so it is not necessary to request an authorization for them.

The Air Navigation Service of the Czech Republic hosts the website "Létejte zodpovědně (Fly Carefully)" about drone problems and has written advisories for hobby drone pilots using clear, understandable language. This website is described in detail in the Related projects chapter 1.6.

Motivation

One motivation for choosing this thesis was to gain experience with mobile application development. I was interested in trying something new, and I had previously only had experience with web development. The other reason was the success in various competitions of both Dronetag's company co-founders, Lukáš Brchl and Marian Hlaváč. They won the Space application hackathon 2018 [5], and they have won multiple competitions on the European and world levels. Finally, I wanted to work with a product with a great deal of potential, and drones are considered a product of the future; business with drones is expected to experience exponential growth.

Objectives

The goal of this thesis is to develop a cross-platform mobile application for safer drone operation. The application will be implemented the best-practice principles using the Flutter framework. In addition, this thesis contains a description of the basic Flutter framework concepts in the Flutter Analysis chapter 3. The implementation contains both testing by unit tests and deployment specifications, too. The application meets all functional requirements, specifically drawing flight zones on the map, showing real-time data about flying drones with a Dronetag device attached, and flight planning and management of aircrafts, devices and flight history.

Chapter 1

Related Projects

This chapter aims to related projects dealing with safe drone operations, especially web and mobile applications, to subscribe to current information about flight restrictions. It involves showing a restricted area to flying, guide with advisories, and terms that a pilot has to meet during his flight.

1.1 AirMap

AirMap is the main competition application comparable with the Dronetag platform. [6] This application shows flight zones and restricted areas around almost the whole of world. Besides, it allows for easy flight planning. Unfortunately, this planning has no central management. Thus this solution is insufficient for ensuring aviation safety. AirMap ecosystem can divide into some branches:

- AirMap UTM,
- AirMap Pilot,
- AirMap Enterprise,
- AirMap Developers.

1.1.1 AirMap UTM

AirMap UTM (Unmanned Traffic Management) is a web application used for aviation flight dispatchers. It allows approving planned flights of users in restricted areas by authorities - for example, CTR (Control zone). The manager of this zone can show a detail of the application form and request that he fill out the information about a flight.

Advantages:

- An operator can see active flight plans and other flight operations and altitudes (airplanes and helicopters) simultaneously.
- An operator can see a DJI drone telemetry of a pilot in case that the pilot is using AirMap mobile application in-flight mode.
- Whenever an operator create a restricted zone for flights, pilots will be notified via SMS.
- An operator can filter received application forms.
- It is used to using in the USA, the Czech Republic, Switzerland, Japan, New Zealand, ...

Disadvantages:

- It offers no on-premise solution.
- It is quite a trivial implementation so far.

1.1.2 AirMap Pilot

The main goal of this application is to force pilots to fly carefully according to their location stringent rules. Concurrently, it provides official authorization to the restricted airspace. Also, the application allows controlling DJI drones instead of the official DJI GO mobile application.

After the launch of the application, the flight map is loaded with map data from Mapbox [7], and a current user location is focused on the map with available zones around. There are immediately seen labels of every zone and advisories for flights in the given country on the map. The suggestion list does not look intuitive. In the bottom panel with advisories is also visible a temperature and wind speed in the location. Unfortunately, it misses various values for arbitrary flight levels. That is it all the functionality for a user, nothing else.

Advantages:

- It is used to using in the USA, the Czech Republic, Switzerland, Japan, New Zealand, ...
- The application shows flight zones of all countries. It loads data from the EUROCONTROL EAD (European AIS Database). [8]
- Pilots can get authorization of airspace via the LAANC (Low Altitude Authorization and Notification Capability) system in real-time.

- Flights can also be planned in the future, and it is not allowed to change them after that. It is allowed only to remove it.
- In case of planning a mission in the web application, it is not able to see it in the mobile application in a minute. However, the mission date does not match. In the same case, when users cancel it in the mobile application, it can be seen it will disappear in the mobile and web application in a minute.
- It contains geofencing alerts - it notifies a pilot about the entrance int the managed flight zone.

Disadvantages:

- In mobile application is missing the option to click on the requested zone and do anything with it or get information. It is quite inconvenient in the case of many intersected zones at one point. The web application allows it.
- Many times it has been got a timeout request, and sometimes something was showing inconsistent. For example, it has not seen any rules in the Flight Briefing section while planning the first flight mission. In another attempt, it has been already seen the rules. Generally, all loading or submitting takes a long time.
- It can provide no flight log export nor other data.
- It supports no flight plan storage for the future nor repeating.
- It contains a few layperson bugs. For example, in the web application, if users draw a trajectory and press ESC key on the keyboard, the users will not be able to draw again without the need to choose a tool and then the default again.

1.1.3 AirMap Enterprise

AirMap Enterprise is a dashboard similar to AirMap UTM, but it is specialized to the narrow group of users. It is not possible to see visible differences against the AirMap UTM, but the screen looks similar. For example, application form approval, real-time telemetry, pilot contacting. It is mainly used as a fleet management system.

1.1.4 AirMap Developers

a AirMap Developers is a portal offering documentation about AirMap API and SDK, that can use for integration of the AirMap services into individual solutions. [9]

1.2 Altitude Angel

Altitude Angel ecosystem can be divided into these three subsystems:

- GuardianUTM,
- Guardian Mobile,
- Drone Safety Map.

1.2.1 GuardianUTM

”GuardianUTM provides software developers and drone manufacturers with the tools and data to access accurate, up-to-date and relevant aeronautical, environmental, regulatory and drone-centric operational data.” [10] This system ensures the integration of flight aviation data in the airspace.

1.2.2 Drone Safety Map

It is a mobile and web application for drone pilots connected to GuardianUTM. Its goal is to show zones and flight missions of all pilots.

Advantages:

- The application shows flight zones in all countries.
- Aside from zones, it also shows hazardous objects like high-voltage wires, schools, police officers, and gas stations. There is much information and it could be visualized more appropriate.
- It supports CAA UK (Civil Aviation Authority of the United Kingdom) that integrates its solution into the application of NATS Drone Assist. [11]

Disadvantages:

- The registration is inconvenient with an additional password setting via e-mail.
- The flight plan history contains unnecessary text fields (flight title, description, and timezone). The flight plan history does not exist.
- There is no chance to share telemetries in real-time (for example, connection with DJI drones).
- It does not allow the planning of more complex zones than circles.
- It is difficult to determine a reserved zone for flight against restricted zone by the authorities in the user interface.

- The mobile and web application has less elaborate user interface beside AirMap. For example, it shows terms & conditions before every flight.
- The mobile application contains bugs in the mission planning and searching.

1.3 MAIA

MAIA is an application by the UpVision company that is primarily used for displaying understandable information about flight zones. [12]

The user interface is mostly consisted of the map that contains national flight zones and restrictions. In case of sufficient data, it will display the location of real-time drones on the map and their details about aircrafts in the airspace.

The signed up users can have a summary of their drones, battery status and flight history. Every user can contact any pilots via the Messenger function, that are visible on the map.

A user can add an insurance confirmation, pilot certification and permission for flight operations in the application. Besides that, users can share their favorite flight locations.

The application of the enterprise version promises functions for drone fleet owners such as geofencing, central management, drone detection and collision detection.

Advantages:

- The application shows positions of drones in real-time.
- The application shows national flight zones and restrictions.
- The application provides data about flight zones in the Czech Republic and other countries.

Disadvantages:

- The web platform has just few functions. Users can be signed in, but it contains no map layers.
- The application requires the MAIA tracker. Otherwise, it does not support another function. [12]
- The mobile application contains many bugs and is not up-to-date.

1.4 Kittyhawk

Kittyhawk is a mobile and web application that has more functionalities than MAIA. All flight operations and rules are similar to AirMap. Also, it offers the fleet management, checklists and battery management. [13]

Advantages:

- It allows limited manner support for DJI drones flights (for example, live stream).
- There are many map data and user-friendly map layer filtering.
- Moving across the map and searching in zones is more user-friendly than in other applications.
- It shows anonymous live traffic on the map.
- There are well-arranged reports, that can be exported.

Disadvantages:

- It displays zones only in the USA, it shows no zones in the EU.
- It loads all data related to flights from AirMap (map data, weather, authorization).
- The registration takes a long time and requires a phone number. It does not support Google account login.
- UI is not entirely intuitive. For example, if users want to plan a flight and have not added a drone yet, they can see only a blank screen and cannot add the flight plan.

1.5 AisView

AisView (or its simplified DroneView) is an application managed bye the Air Navigation Service of the Czech Republic before starting flight purposes. There are restrictions, map zones, NOTAMs (Notice To Airmen), and planned drone flights on the map. It has no real-time drone location data.

Users set flight a time range (from and to), and after they will be able to see relevant information related to them in the map during the time range. Besides, the users must manually click to shown zones to see more information.

The users can see a map covered with additional information about the weather (especially a layer of data from CHMI radar [14]) or choose a type of map data (ICAO [15], orthographic, tourist).

The user interface is quite unfriendly and non-intuitive in some cases. Abbreviations sometimes label the buttons, and, certainly, the application is rather for professionals. A novice can have problems with this application and does not have to understand its possibilities. The DronView does not support such usable functionalities for users than the AisView.

After logging in, users can add registration numbers of their devices and announce drone flight plans. Also, there are options to store various minor showing map preferences and other details for the logged users.

AisView primarily provides especially a web variant, that can use on the computer and mobile phones. There is a variant for mobile phones, too. However, it is quite limited and looks obsolete and does not include all system functions.

Advantages:

- It provides data from the verified source.
- The application is used by pilots in the daily routine and is stable.
- A user can announce his flight plan to authorities from the application.
- It is the main source of the Air Navigation Service of the Czech Republic, so it is always up-to-date.

Disadvantages:

- The application user interface looks obsolete, and the user experience is often unsatisfactory for the public.
- It can see planned drone flights, except real-time drone positions in the application.
- The geographic range is only for the Czech Republic. The data of other countries are not visible.

1.6 Fly Carefully (Létejte zodpovědně)

Fly Carefully is an educational website whose main purpose is to educate hobby drone pilots and gets them to know with valid regulation. Rules for flying with drones describe precisely and offer help to total beginners. This website is managed by the Air Navigation Service of the Czech Republic what is the state organization ensuring aviation safety. [16] It is divided into 7 basic sections:

1. **Introduction** – it contains important advisories for flights,
2. **My drone** – it is a section about drone categories and types,

3. **Plan your flight** – it gets to know readers about options to plan a flight,
4. **Our application** – it introduces a secondary product what a mobile application is,
5. **Regulation** – it informs readers about valid flight regulations and advisories,
6. **Actualities** – it represents news feeds,
7. **Registration** – it provides a form to subscribe to newsletters.

This mobile application is similar to the AirMap application because it uses the AirMap public API and it offers nothing new against the AirMap mobile application.

Chapter 2

Dronetag Web Infrastructure

This chapter describes the Dronetag web infrastructure and contains a detailed description of the Dronetag web platform. It consists of the all technical stack that ensures data providing and processing. It was needed to determine a convenient and reliable way of how to construct a useful architecture of infrastructure gradually. The Dronetag development lifecycle has started with an IoT module and frontend web client written in JavaScript. It continues to implement Web Sockets in the Live Service implementation based on a Kubernetes cluster and the cross-platform mobile application. Current parts of the stack are the following:

- **Load Balancer** – it ensures load balancing of received requests (It redirects the requests to an available node in cluster.),
- **Backend** – it ensures communication with Database storage of data and provides API endpoints to allow drone security operation,
- **Frontend** – it ensures managing and watching drones activity via a web browser,
- **Private API endpoint** – it ensures an API endpoint to give into or get private data from Dronetag platform (It is a part of Backend.),
- **Public API endpoint** – it ensures an API endpoint to give into or get public data from Dronetag platform (It is also a part of Backend.),
- **Database Storage** – it ensures persistent storing of data,
- **Live Service** – it ensures quick providing of live data from airspace by the given viewport.

It is needed to note the stack is still improved anytime when there is found out a problem with the insufficient and complexity of the all system. It is needed because the system complexity is still increasing. And in this case, it is needed to determine the cause and ensure load balancing among the parts of the stack.

2.1 API Separation

Due to security reasons, the web API- divides itself into Private API and Public API endpoints. Public API is allowed to use by third-party consumers. It can be anyone who would like using data from the Backend to ensure drone operation security.

In further determination, this infrastructure is divided into the Staging and Production environment. Staging is for development and testing purposes, and it usually runs on the same version as Production. In the infrastructure approach, Production is like a mirror of Staging. The difference is that the Production environment contains a previous version of the application that is released.

2.2 Docker Deployment

Every web application in the stack is deployed and managed by Docker. It is the easiest way to develop and publish new version software. However, in the beginning, let us clarify what the Docker is. ”Docker is a tool designed to make it easier to create, deploy, and run applications by using containers. Containers allow a developer to package up an application with all of the parts it needs, such as libraries and other dependencies, and deploy it as one package. By doing so, thanks to the container, the developer can rest assured that the application will run on any other Linux machine regardless of any customized settings that machine might have that could differ from the machine used for writing and testing the code.” [17] To read about what a Container is and how useful it can be for business, it is able to see on their official websites [18] or here [17].

Thanks to the Docker Compose tool, it is able to separate the infrastructure into parts. In the case of more significant changes, it is needed to change only one module, and others are without changes. In a summary, it deploys more containers that each of them contains individual responsibility for their processing, and Docker Compose joins them together. In abbreviation, Docker Compose works like a composer that takes separate containers, where each of them can launch different technologies and compose it together into one whole infrastructure. [19]

2.3 Kubernetes

”Kubernetes (K8s) is an open-source system for automating deployment, scaling, and management of containerized applications.” [20] It means that Kubernetes is a tool allowing a running cluster whose single nodes may be as Docker containers. This approach ensures scalability, whereas the number of Docker containers can be huge and can be created and disposed of dynamically based on the loading. The advantage of Kubernetes is that it can detect a note out of order and initialize and deploy a new one to stay the optimal performance

”It groups containers that make up an application into logical units for easy management and discovery. Kubernetes builds upon *15 years of experience of running production workloads at Google* [21], combined with best-of-breed ideas and practices from the community.” [20] Thanks to these experiences, Kubernetes is a fine grain, and the cluster hierarchy is merely maintainable.

”Though widespread interest in software containers is a relatively recent phenomenon, at Google we have been managing Linux containers at scale for more than ten years and built three different container-management systems in that time.” [21] How you can see, the idea of containerization has a rich history. However, there were Linux Containers before the Docker ones. That is the reason why we use it for web development. It is easy to scale, maintain, and able to deploy to Google Cloud Platform.

2.4 Database Model

There is a database model [22] to store data in Dronetag Backend. The database model consists of followings entities:

- User,
- Aircraft,
- Device,
- Flight,
- Telemetry measurement,
- Organization,
- Fleet,
- Aircraft vendor,
- Aircraft model,
- Airspace zone,
- User preference.

2.4.1 User

This entity represents a user who signs up and logs in to the application. It consists of user credentials and identification information. Attributes of this entity are the following:

- **E-mail** – represents the e-mail that the user will be log in to the system and receive notifications,
- **Full name** – represents optional full name of the user,
- **Password hash** – represents a hashed password,
- **Phone number** – represents a phone number that the user will be able to contact in case of emergency,
- **Country** – represents a country where the user is used to fly.

2.4.2 Aircraft

This entity represents an aircraft that is connected with a Dronetag device. Attributes of this entity are the following:

- **Name** – represents the aircraft name for easier recognition in My aircraft list,
- **UAS Operator ID** – represents a unique code identifying a pilot who registered this aircraft in the UAS (Unmanned aircraft systems) [23],
- **Weight** – represents the weight of the aircraft.

2.4.3 Device

This entity represents a physical device that sends live information to the Dronetag platform. Attributes of this entity are the following:

- **Serial number** – represents a serial number of the device,
- **Name** – represents the device name for easier recognition in My aircraft list,
- **Type** – represents the model type of the device,
- **Last battery** – represents the last battery value in Volts,
- **Last RSRP** – represents the RSRP (Reference Signal Receive Power) value.

2.4.4 Flight

This entity represents it represents a flight that a user has created. Attributes of this entity are the following:

- **Date planned start** – represents a start date of planned the flight,
- **Date planned finish** – represents a finish data of planned the flight,
- **Date started** – represents a real start date of the flight,
- **Date finished** – represents a real finish date of the flight,
- **Status** – represents a flight status - values can be planned, current, finished and canceled,
- **Distance** – represents a distance of the flight,
- **Duration** – represents a duration of the flight,
- **Region GeoJSON** – represents a reservation region in GeoJSON format [24],
- **Max flight altitude** – represents a maximum flight altitude,
- **Takeoff latitude** – represents a latitude of a taking off position,
- **Takeoff longitude** – represents a longitude of a taking off position,
- **Takeoff Geo altitude** – represents a geological altitude of a taking off position,
- **Takeoff pressure** – represents a taking off pressure,
- **Public** – represents a boolean flag if the flight is public (visible for everyone).

2.4.5 Telemetry Measurement

This entity represents a telemetry measurement that the system receives and thanks so that a flight trajectory can be drawn. Attributes of this entity are the following:

- **Time** – represents a date with the time of measurement,
- **Latitude** – represents a measured latitude of a flying drone,
- **Longitude** – represents a measured longitude of a flying drone,
- **Altitude** – represents a measured altitude of a flying drone,

- **Geo altitude** – represents a measured geological altitude of a flying drone,
- **Velocity X** – represents a measured velocity in X-axis,
- **Velocity Y** – represents a measured velocity in Y-axis,
- **Velocity Z** – represents a measured velocity in Z-axis.

2.4.6 Organization

This entity represents an Organization for the fleet management functionality. Attributes of this entity are the following:

- **Name** – represents the name of the organization,
- **Description** – represents a text description of the organization.

2.4.7 Fleet

This entity represents an organization's fleet management properties. Attributes of this entity are the following:

- **Name** – represents a name of the fleet,
- **Color** – represents a shown color of the fleet,
- **Deleted** – represents a boolean flag if the fleet was deleted.

2.4.8 Aircraft Vendor

This entity represents an aircraft vendor who manufactures drones. It contains only the one attribute name that represents the vendor name.

2.4.9 Aircraft Model

This entity represents an aircraft model that belongs to a vendor. Attributes of this entity are the following:

- **Name** – represents the model name,
- **Weight** – represents the weight of the model,
- **Vendor ID** – represents a relationship to a Vendor.

2.4.10 Airspace Zone

This entity represents an airspace zone. Attributes of this entity are the following:

- **Name** – represents a name of the zone,
- **Country** – represents a country where the zone belongs,
- **Region GeoJSON** – represents a region in GeoJSON format [24].

2.4.11 User Preference

This entity represents a user preference that is needed to store and share among various client applications. Attributes of this entity are the following:

- **Property** – represents an identification of the property,
- **Value** – represents the preference value.

2.4.12 Live Service Database Model

During the development, it had been found out the current Backend is not sufficient for Dronetag needs. So it was decided to divide the backend model into the Backend and Live Service model. The Live Service contains only live real-time temporary data, so a Redis database was deployed.

”Redis is an open source (BSD licensed), in-memory data structure store, used as a database, cache and message broker. It supports data structures such as strings, hashes, lists, sets, sorted sets with range queries, bitmaps, hyperloglogs, geospatial indexes with radius queries and streams. Redis has built-in replication, Lua scripting, LRU eviction, transactions and different levels of on-disk persistence, and provides high availability via Redis Sentinel and automatic partitioning with Redis Cluster.” [25] It means that the Redis is real-time storage that persists data only for a short time. That is the reason why it is suitable for this purpose. The Live Service database model consists of a Device and Telemetry entity.

Chapter 3

Flutter Analysis

This chapter describes the key reasons why Dronetag chose Flutter for mobile application development.

Nowadays, many companies have more teams to maintain their application that ensure the business of these companies. Thanks to that, the costs are higher than could be, and that is the reason why cross-platform frameworks were created. There are many cross-platform mobile application platforms to develop an application like this. The name of the leading used frameworks are:

- Flutter,
- React Native,
- Ionic,
- Xamarin.

3.1 Cross-platform Mobile Application Framework Comparison

This section describes a comparison of cross-platform mobile application frameworks. These frameworks are Flutter, React Native, Ionic and Xamarin.

Flutter is a framework and was established by Google Inc., in 2017, and since that time, its popularity is still increasing. React Native looks like the leading competitor and is based on JavaScript. [26] It means, React Native is widespread and most popular. Ionic has not such excellent performance as Flutter and React Native. [27] Xamarin is based on C# with .NET extension from Microsoft. But since there is a lack of experienced Xamarin developers in the developer community, it was omitted in the Dronetag's decision making. Flutter is based on Dart, which was introduced by Google Inc., in 2011. It is a type-safe and object-oriented programming language similar to Java. [28]

Among the advantages of Flutter [29] belongs:

- Hot Reload, i.e., allows fast coding,
- One codebase: Development for two mobile platforms,
- Up to 50 % less testing,
- Faster app development,
- User-friendly designs,
- Perfect for MVPs,
- Less Code.

There is a comparison summary:

- Flutter has the best performance and uses proprietary Widgets to create a user interface instead of HTML, CSS and JavaScript.
- React Native is the most popular because it is based on JavaScript.
- 98 % of code in Ionic is reusable.
- Xamarin allows reusing code in Xamarin.Forms, but requires a paid development environment for business purposes.

3.2 General Concept

”Flutter is an app SDK for building high-performance, high-fidelity apps for iOS, Android, web (beta), and desktop (technical preview) from a single codebase.” [1] It means you need only one development team to reach an application for all platforms in a single codebase.

”The goal is to enable developers to deliver high-performance apps that feel natural on different platforms. We embrace differences in scrolling behaviors, typography, icons, and more.” [1]

Thanks to this, it is able to style the application user interface on demand, and the application looks similar to both platforms.

3.3 User Interface in Flutter

”Flutter includes a modern react-style framework, a 2D rendering engine, ready-made widgets, and development tools. These components work together to help you design, build, test, and debug apps. Everything is organized around a few core principles.” [1] Flutter has a widget tree that renders widget

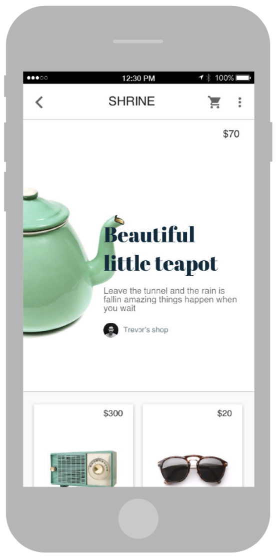

SHRINE
$70
Beautiful little teapot
Leave the tunnel and the rain is fallin amazing things happen when you wait
Trevor's shop
$300
$20

Figure 3.1: Flutter demo app on iOS [1]

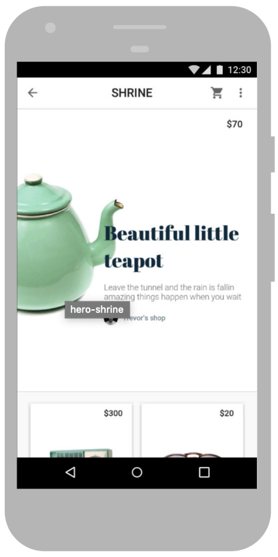

SHRINE
$70
Beautiful little teapot
Leave the tunnel and the rain is fallin amazing things happen when you wait
hero-shrine
Trevor's shop
$300
$20

Figure 3.2: Flutter demo app on Android [1]

into a nested tree, and these widgets are covering themselves. It does not matter if a widget from the widget tree is:

- Container,
- List View,
- Image,
- Text,
- Animation,
- or anything else.

A widget or its descendants represent everything in the user interface in Flutter. Additional important classes for user interface are MaterialApp and Scaffold. [30] MaterialApp and Scaffold, it will be described in the Material Design section 3.8.

3.4 Widgets

How the founders of Flutter say: ”Everything is a Widget”. In the following part, this statement will be clarified.

”Widgets are the basic building blocks of a Flutter app’s user interface. Each widget is an immutable declaration of part of the user interface. Unlike other frameworks that separate views, view controllers, layouts, and other properties, Flutter has a consistent, unified object model: the widget. A widget can define:

- A structural element (like a button or menu),
- A stylistic element (like a font or color scheme),
- An aspect of layout (like padding),
- And so on ...” [1]

So each component of UI in Flutter is a descendent of the widget. There are many classes that inherit from a widget. The essential descendants are the following:

- StatelessWidget,
- StatefulWidget,
- InheritedWidget.

3.4.1 StatelessWidget

”A widget that does not require mutable state. A stateless widget is a widget that describes part of the user interface by building a constellation of other widgets that describe the user interface more concretely. The building process continues recursively until the description of the user interface is fully concrete (e.g., consists entirely of *RenderObjectWidgets* [31], which describe concrete *RenderObjects* [32]).” [33] It means that StatelessWidget is a useful component to render graphics elements that do not change during their lives. StatelessWidget is class with Constructor and build method. When the widget tree is built, it calls the build method.

”Stateless widget are useful when the part of the user interface you are describing does not depend on anything other than the configuration information in the object itself and the BuildContext in which the widget is inflated. For compositions that can change dynamically, e.g. due to having an internal clock-driven state, or depending on some system state, consider using StatefulWidget.” [33]

3.4.2 StatefulWidget

”A widget that has mutable state. State is information that (1) can be read synchronously when the widget is built and (2) might change during the lifetime of the widget. It is the responsibility of the widget implementer to ensure that the *State* [34] is promptly notified when such state changes, using *State.setState* [35].” [36] It means that StatefulWidget is a useful component to render graphics elements that change until they are disposed of. StatefulWidget initializes its state, which represents a space for storing data. The State is class with constructor, initState and build method. The constructor is called when the State is created. The initState method is called before the widget tree is rendered, and the build method is called when the tree renders itself.

Due to this fact, the Stateful widget has worse performance than StatelessWidget, and often it is difficult to keep a sustainable design of a component. On the other hand, it offers a convenient way to create a widget with few states that will not be changed in the future development cycle.

”Stateful widgets are useful when the part of the user interface you are describing can change dynamically, e.g. due to having an internal clock-driven state, or depending on some system state. For compositions that depend only on the configuration information in the object itself and the BuildContext in which the widget is inflated, consider using StatelessWidget.” [36]

3.4.3 InheritedWidget

”Base class for widgets that efficiently propagate information down the tree.

To obtain the nearest instance of a particular type of inherited widget from a build context, use *BuildContext.dependOnInheritedWidgetOfExactType* [37].

Inherited widgets, when referenced in this way, will cause the consumer to rebuild when the inherited widget itself changes state.” [38] It means that InheritedWidget is a useful component to pass data and offer to reduce boilerplate if there are many widgets nested in themselves. Thanks to the BuildContext class and of a method, you can easily get the value you add as an input variable. So it means the widget is suitable for passing a huge amount of data. It loses a need to copy many parameters in a large tree structure.

3.5 Bloc

Bloc is an abbreviation of Business Logic Component (BLoC) and allows separating an application into separate layers. [2] ”The BLoC Pattern has been designed by Paolo Soares and Cong Hui, from Google and first presented during the DartConf 2018 (January 23–24, 2018).” [39]

”The goal of this package is to make it easy to implement the BLoC Design Pattern.

This design pattern helps to separate presentation from business logic. Following the BLoC pattern facilitates testability and reusability. This package abstracts reactive aspects of the pattern allowing developers to focus on converting events into states.” [2]

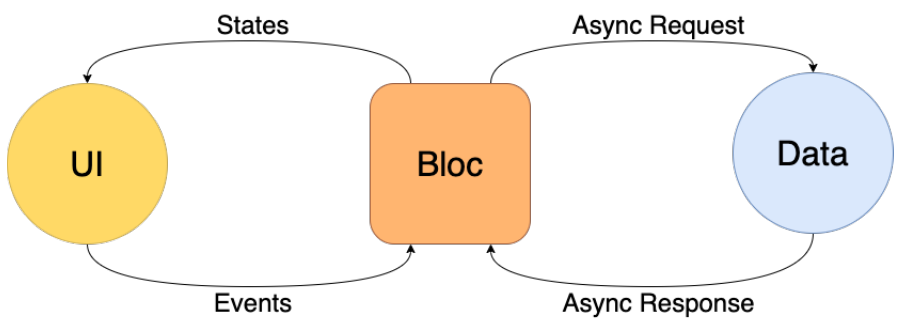


Figure 3.3: Bloc architecture [2]

BLoC is a library written by Felix Angelov and the concept bases on Reactive Programming. [2] BLoC represents a control unit that is responsible for passing data from state into UI. When a user clicks on a button, it throws an event action that BLoC detects and generates a new state. [2] In a summary, a structure of the concept consists of the following parts in Bloc:

- **Events** are the input to a BLoC. They are commonly UI events such as button presses. Events are added to the BLoC and then converted to States.
- **States** are the output of a BLoC. Presentation components can listen to the stream of states and redraw portions of themselves based on the given state (see BlocBuilder for more details).
- **Transitions** occur when an Event is added after mapEventToState has been called but before the BLoC's state has been updated. A Transition consists of the currentState, the event which was added, and the nextState.
- **BlocSupervisor** oversees BLoCs and delegates to BlocDelegate.
- **BlocDelegate** handles events from all BLoCs which are delegated by the BlocSupervisor. Can be used to intercept all BLoC events, transitions, and errors. It is a great way to handle logging/analytics as well as error handling universally.

3.6 HydratedBloc

"*hydrated_bloc* [40] is an extension to the *bloc state management library* [2] which automatically persists and restores bloc states.

hydrated_bloc exports a HydratedStorage interface which means it can work with any storage provider. Out of the box, it comes with its own implementation: HydratedBlocStorage.

HydratedBlocStorage is built on top of *path_provider* [41] for a platform-agnostic storage layer. The out-of-the-box storage implementation reads/writes to file using the **toJson/fromJson** methods on HydratedBloc and should perform very well for most use-cases (performance reports coming soon). HydratedBlocStorage is supported for desktop (*example* [42])." [43]

HydratedBloc works as well as common Bloc. The difference is in data storage. Hydrated Bloc allows us to store data through a JSON object. So, whenever an application loads data, it is necessary to wait and show the user progress indicator, but it can show stored data immediately. When it receives data, it will simply render it into a screen.

"HydratedBlocStorage is an implementation of *HydratedStorage* which uses *PathProvider* and *dart.io* to persist and retrieve state changes from the local device." [44] So, it is a part of Hydrated Bloc implementation.

3.7 Cupertino Library

The Flutter developers have decided to incorporate familiar elements from the iOS platform into the library for using these elements in the Flutter user interface. The reason is that it would be easy to use for development, focusing on iOS devices. This library calls Cupertino and the name was established by the Apple company headquarters building in Silicon Valley, California, in the United States of America. Familiar elements from iOS devices are part of the library because the iOS users are using them. [45] This library contains the following elements:

- Cupertino Action Sheet,
- Cupertino Activity Indicator,
- Cupertino Alert Dialog,
- Cupertino Button,
- Cupertino Context Menu,
- Cupertino Date Picker,
- Cupertino Dialog,
- Cupertino Navigation Bar,
- Cupertino Page Scaffold,
- Cupertino Picker,
- Cupertino Slider,
- Cupertino Switch,
- Cupertino Tab Scaffold.

3.8 Material Design

Material Design is a concept that was introduced by Google company in 2014. [46] All Flutter user interface is built on this concept. Material Design contains components that interactively build blocks for creating a user interface. [47] The components are the following:

- App bar,
- Bottom navigation,
- Buttons,
- Floating Action button,
- Cards,
- Chips,
- Dialogs,
- Lists,
- Pickers,
- Progress indicators,
- Sliders,
- Tabs.

3.8.1 Scaffold

Scaffold represents a new rendered screen that allows placing various elements. The Scaffold is possibly fully customized and change by the requirements. If developers emphasize simplicity, it is able to use the current interface and define:

- AppBar,
- FloatingActionButton,
- BottomNavigationBar.

AppBar represents the header of the screen and usually contains a title and action buttons. FloatingActionButton is a concept of buttons that users use the most time, and thus these buttons are in the thumb zone. BottomNavigationBar is a concept of a horizontal menu in the thumb zone in the footer of the screen, and it allows the creation of the main well-arranged menu. [30]

These two libraries are interesting for this mobile application. So, these elements are being used from these libraries in the user interface by the platform on the phone, where the application is running. The reason for using is simplicity. The users should get what they like and are used to use.

Chapter 4

Software Architecture and Design

This chapter describes the software architecture and design of the mobile application. In the beginning, it will be clarified the difference between software architecture and design.

In software development, software architecture means the design of application from a higher perspective than software design. [48] In opposite, software design means the division of the code into parts that describe implementation details accurately. [49] It means that when anyone starts with a new application project, it is essential to arrange the project into small parts. These parts should meet the encapsulation and comprehensive rules.

Architecture usually means a division into more significant self-driven parts. There are typical concepts, MVC (Model-View-Controller), and their extended descendants, MVP (Model-View-Presenter) and MVVM (Model-View-ViewModel). [50] The newer great concept in Flutter is BLoC. It has been already explained what the Bloc is in the Flutter Analysis chapter 3, so it will not be clarified anymore.

4.1 Software Architecture

This section focuses on the architecture of this application implementation. Because the definition of software architecture has been already clarified, it will be emphasized to certain components that were used in the implementation.

In software architecture, there has been verified a pattern of using libraries that offers using of generic classes. These classes include already implemented logic that is based on standards, and thus it helps to simplify development and maintain code more readable. For better understanding, Bloc and Hydrated Bloc classes will be described in the section 4.1.2 of this chapter.

There would be useful to discuss the difference between common Bloc and its extended class Hydrated Bloc. Both of them use the sink and stream concept that works on the stream subscription principle. The Hydrated Bloc offers persistent storage storing and loading with **fromJson** and **toJson** methods. It helps to have an application working more smoothly due to storing of data in the phone. Only continuous loading data via the network whenever a user needs them has significant impact to the response time in the application.

4.1.1 Dependencies

Flutter has a vast database of dependencies that are maintained by various developers. The database is called pub.dev [51] and offers a summary of every dependency on the score analysis to their users. This analysis consists of popularity, health, maintenance and overall.

The advantage of these dependencies is that they can be easily integrated. Developers can place their definition into **pubspec.yaml** file that describes the dependency configuration. The file is placed in the root directory of every Flutter project. The definition consists of name and specific version.

The dependencies that are used in this mobile application are separated into the following categories:

- Icons,
- State management,
- Tools,
- Firebase,
- API,
- NoSQL storage,
- UI,
- Map,
- Generator.

Icons contain **cupertino_icons** library that provides iOS Cupertino Icons by Apple Inc. Material Design icons are already part of the Flutter SDK library.

State management consists of **flutter_bloc** and **hydrated_bloc**. These dependencies contain the Bloc and Hydrated Bloc implementation. The implementation is described in the Bloc section 3.5 of the Flutter Analysis chapter. The usage of the implementation is described in the Bloc and Hydrated Bloc section 4.1.2 of this chapter.

Tools consist of:

- **intl** – it ensures internationalization,
- **equatable** – it ensures that object extending it can be easily comparable by properties of that object,
- **flutter_svg** – it ensures drawing SVG image as a widget,
- **flutter_i18n** – it ensures internationalization user phone settings,
- **path_provider** – it ensures finding commonly used locations the filesystem,
- **print_lumberdash** – it ensures a convenient debug printing of an error message,
- **get_it** – it allows using of DI (Dependency Injection) in code - it supports IoC principle, [52]
- **dartz** – it allows using functional programming in Dart,
- **url_launcher** – it ensures launching a URL in the mobile platform,

Firebase contains **firebase_core** and **firebase_messaging**. These dependecies ensure connecting to Firebase Cloud and allow push notification functionality.

API consists of:

- **http** – it ensures making HTTP requests,
- **json_annotation** – it ensures the definition of annotations used by JSON serialization and deserialization,
- **data_connection_checker** – it checks for an internet connection by opening a socket,
- **dio** – it offers support of Interceptors, Global configuration, FormData, Request Cancellation, File downloading, Timeout, etc.,
- **retrofit** - it ensures type conversion dio client generator to create HTTP requests,
- **pretty_dio_logger** – it ensures a convenient debug printing of HTTP requests and responses,
- **flutter_screenutil** – it ensures adapting screen and font size.

NoSQL storage contains **hive** and **hive_flutter**. These dependencies ensure NoSQL storage that is for storing persistent data in the offline mode, JWT token and user preferences. [53]

UI consists of:

- **sliding_up_panel** – it offers using done Sliding up panel,
- **animations** – it contains pre-canned animations for commonly-desired effects,
- **flutter_sfsymbols** – it supports all iOS and Android devices.

Map consists of:

- **google_maps_flutter** – it provides a Google Maps widget,
- **location** – it handles getting a location on Android and iOS,
- **google_maps_webservice** – it offers using Google Web Service API that including Geocoding, Places, Directions, and so on.

Generator consists of:

- **hive_generator** – it generates storable object by Hive annotations,
- **retrofit_generator** – it generates an implementation of HTTP requests due to set annotations,
- **bloc_test** – it generates testing blocs for unit testing,
- **mockito** – it generates Mocks for unit testing,
- **pedantic** – it ensures a linter that is looking after to meet best practices,
- **build_runner** – it provides a concrete way of generating files using Dart code,
- **json_serializable** – it generates a serializable object by JSON annotation.

4.1.2 Bloc and Hydrated Bloc

In the beginning of the development, all BLoC (Business Logic Component)s were the descendants of the Bloc class. After a short time, it has been found out that the Hydrated Bloc pattern is more convenient for this purpose. Therefore, some Bloc classes were changed to descendants of HydratedBloc. This is a list of classes that has been created:

- User Profile Bloc,

- Authentication Bloc,
- Aircraft Operation Bloc,
- Device Operation Bloc,
- Flight Operation Bloc,
- Device Detail Bloc,
- Location Bloc,
- Map Bloc,
- Map Zone Bloc,
- Map Drone Bloc,
- Map Place Bloc.

In the beginning, **User Profile Bloc** extended common Bloc, but after increasing data traffic, it has changed to Hydrated Bloc. The reason is simple. User properties are not changed so often. Due to it is better to store the data on the phone and update them when it changes. User Profile bloc ensures loading and updating user data.
Authentication Bloc extends Bloc and ensures:

- Logging in a user,
- Logging out user,
- Finding out if a user is logged in.

Aircraft Operation Bloc extends Hydrated Bloc and ensures:

- Loading aircraft data,
- Adding new aircraft,
- Updating an aircraft,
- Deleting an aircraft.

Device Operation Bloc extends Hydrated Bloc and ensures:

- Loading device data,
- Registration of a device to a user,
- Updating a device,
- Deleting a device.

Flight Operation Bloc extends Hydrated Bloc and ensures:

- Loading flight statistics data,
- Loading flights data,
- Loading flight data,
- Updating a flight,
- Deleting a flight.

Device Detail Bloc extends Bloc and ensures:

- Switching from a Battery icon to a value in Volts, and vice versa,
- Switching from a LTE icon to a RSRP value id dBs, and vice versa,
- Switching from a Bluetooth icon to a RSRP value in dBs, and vice versa.

This Bloc is used for switching of icons in the Device detail screen.

Location Bloc extends Hydrated Bloc and ensures focusing to a user location by GPS coordinates in the Dashboard map.

Map Bloc extends Bloc. Map Bloc is the most complex component in the whole project. It looks after all user interaction with the Dashboard map. Map Bloc mediates:

- Loading zones,
- Finding all zones covering a given point,
- Showing and closing zone selection,
- Showing and closing zone detail,
- Loading flying drones,
- Showing and closing drone detail,
- Focusing on a drone location,
- Showing and closing place detail,
- Focusing to a place location,
- Pinning and unpinning place marker into the map,
- Storing map object like Markers, Circles, Polygons and Polylines.

In addition, it has nested Blocs that listen the State changes. These Blocs are the following:

- Map Zone Bloc,
- Map Drone Bloc,
- Map Place Bloc.

These Blocs react on the Map Bloc State changes, which means when a new State is created, a nested Bloc catches it and generates a new Event. The Event ensures the real action on the level of the nested Bloc thanks to mapping methods.

Map Zone Bloc extends Bloc and ensures:

- Loading zones,
- Finding all zones covering a given point,
- Showing and closing zone selection,
- Showing and closing zone detail.

Map Drone Bloc extends Bloc and ensures:

- Loading flying drones,
- Showing and closing drone detail,
- Focusing on a drone location.

Map Place Bloc extends Bloc and ensures:

- Showing and closing place detail,
- Focusing on a place location,
- Pinning and unpinning place marker into the map.

4.1.3 JsonSerializable

Json Serializable interface offers generating methods for serialization and deserialization classes. Json Annotation annotates these classes. These classes are converted, and the process creates the methods via the build runner library. The advantage is that developers have not to code these methods by hand. They can re-generated every time it is needed to change the class structure.

Request classes are used for store and serialization of data before the Rest client will send it. This directory contains only Login Request and Register Request classes.

Response classes are used for store and deserialization of data after the Rest client has received it. This directory contains:

- **Aircraft Detail Response** – is used for passing aircraft data in Aircraft Repository,
- **Aircraft Overview Response** – is used for a nested object in Flight Info Response in Flight Repository,
- **Api Error** – is used for processing of API error message in each of Repositories,
- **Device Detail Response** – is used for passing device data in Device Repository,
- **Device Live Response** – is used for passing device live data in Device Repository,
- **Device Telemetry Response** – is used for passing device telemetry live data in Device Repository,
- **Dronetag Overview Response** – is used for a nested object in Flight Info Response in Flight Repository,
- **Flight Info Response** – is used for passing flight data in Flight Repository,
- **Flight Statistics Response** – is used for flight statistics data in Flight Repository,
- **Location** – is used for passing latitude and longitude data in almost each of Repositories,
- **Region Response** – is used for a nested object in Zone Response and Flight Info Response,
- **Take Off Response** – is used for a nested object in Flight Info Response in Flight Repository,
- **Telemetry Response** – is used for passing telemetry data in Device Repository,
- **User Profile Response** – is used for passing user data in User Repository,
- **Zone Response** – is used for passing zone data in Zone Repository.

4.1.4 Dio

Due to the need to use Interception and fill prepared HTTP requests with Authentication Bearer, it was defined the custom Dronetag Dio class. It initializes Pretty Dio Logger to make a clear debug printing and Token Interceptor that extends Interceptors Wrapper.

4.2 Software Design

This section contains the implementation of mobile application design. Because we have already clarified what software design is, this section contains only the list of components that were designed in the mobile application.

4.2.1 Common

This directory is used to maintain smaller parts in the codebase, that are usually shared among many other components. The components that the common directory uses are the elements of the user interface. The directory contains:

- **Common dialog** – it represents an abstraction of Dialog, that receives a template of arbitrary content, that will show as a part of the dialog,
- **Fade In Transition** – it represents a transition when a user goes through among screens in the application,
- **Colors** – it contains named colors for better work and comfort improvement in development,
- **Constants** – it contains named constants that are used for the configuration,
- **Keys** – it contains keys that are needed for a widget identification when an animation runs,
- **Padding** – it contains named sizes of padding for better work and comfort improvement in development,
- **Pair** – it is a class for passing two different objects - it used for returning different data from methods,
- **Size** – it contains named sizes for better work and comfort improvement in development,
- **Theme** – it contains a theme definition via Theme Data class from Material Design.

4.2.2 IO

IO directory contains components that ensure loading data from remote sources. Here it is emphasized to the encapsulation and comprehension. Thus, it has been created a single directory that is divided into, Model, Repositories and Services directories. These nested directories are described in more detail in the following paragraphs.

Model directory contains objects that are produced by API endpoint of Dronetag backend. It is divided into request and response classes. Model contains entities that are used for data objects So, their primary purpose is to keep data consistently. Serialization ensures the facility mapping data to objects that the application gets from the remote data sources. Serialization ensures the interface JsonSerialization that generates serialized objects by the given configuration. This configuration is processed via JsonAnnotation interface in the library with the same name.

Repositories directory consists of a repository for each functional part. Every part contains methods for laboring with individual entities and mediates passing data from remote sources. The part of every repository is a service that ensures proper operations that will process with the given entity from the Model directory. The services are closely described in the next paragraph in this section. The Repositories directory contains the following classes:

- User Repository,
- Aircraft Repository,
- Device Repository,
- Flight Repository,
- Search Repository,
- Zone Repository.

Services directory contains a subdirectory **api**. This subdirectory contains classes that ensure receiving from and sending data to API endpoints that are part of the Dronetag infrastructure. The Backend instance and Live Service ensure the API interface in this infrastructure. Thus, **api** directory is divided into:-

- Dronetag Web Rest Client,
- Dronetag Live Service Client,
- Auth Config,
- Google API Key Reader,
- Dronetag Dio.

Dronetag Web Rest Client is used for loading and sending data o the web platform, specifically the backend. This client contains methods:

- **fetchMyUserProfile()** – it returns a user profile data,
- **authorize(String email, String password)** – it authorizes a user via a JWT token endpoint,
- **registerUser(RegisterRequest request)** – it registers a new user,
- **fetchFlightInfo(String id)** – it returns a flight data by the given id,
- **fetchAircrafts()** – it returns all aircrafts that a user has granted access to see them,
- **fetchAircraft(String uuid)** – it returns an aircraft detail by the given id,
- **addAircraft(String name, String uasOperatorID, String modelID, double weight)** – it adds a new aircraft by the given parameters,
- **updateAircraft(String uuid, String name, String modelID, double weight)** – it updates an aircraft detail by the given uuid,
- **deleteAircraft(String uuid)** – it deletes an aircraft by the given uuid,
- **fetchDevices()** – it returns all devices that a user has granted access to see them,
- **fetchDevice(String uuid)** – it returns a device detail by the given uuid,
- **addDevice(String name, String serialNumber)** – it adds a new device with the given parameters,
- **deleteDevice(String uuid)** – it deletes a device by the given uuid,
- **updateUserProfile(String uuid, UserProfile user)** – it updates a user detail by the given uuid with the user parameter,
- **fetchMyFlightStat()** – it returns user's flight statistics,
- **deleteFlight(String uuid)** – it deletes a flight by the given uuid,
- **fetchZones(String viewport)** – it returns all zones by the given viewport,
- **fetchZone(String uuid)** – it returns a zone detail by the given uuid,
- **passwordReset(String email)** – it resets a user's password by the given e-mail address.

Dronetag Live Service Client is used for loading live data from Live Service. The reason to divide the Rest client into two clients is well-arrangement. The reason is that the web infrastructure is separated into these two services, as described in the Dronetag web infrastructure chapter 2. This client contains methods:

- **fetchAllTelemetries()** – it returns all last live telemetries about all Drones in the airspace,
- **fetchDeviceDetail(String serialNumber)** – it returns a device detail by a given serial number,
- **fetchTelemetryDetail(String serialNumber)** – it returns all live telemetries about a drone by a given serial number.

Auth Config, **Google API Key Reader** and **Dronetag Dio** ensure another mechanism and are used to provide data from the remote sources directly. However, it was suitable to keep these classes together with the Rest clients, because they are closely related to each other. Auth config contains methods:

- **initialize()** – it loads auth_config.json configuration file and stores the values from it by the clientId and clientSecret keys in the Map (data container as similar as List),
- **getClientId()** – it returns a value from the Map container by the *clientId* key,
- **getClientSecret()** – it returns a value from the Map container by the *clientSecret* key.

Google API Key Reader contains methods *initialize()* and *getApiKey()*. *initialize()* loads auth_config.json configuration file and store the values from it by the clientId and clientSecret keys in the Map (data container as similar as List). *getApiKey()* returns a value from the Map container by the *apiKey* key.

4.2.3 UI

This directory is very extensive and every subdirectory contains other subdirectories. Hence, this part describes only a brief description.

Aircrafts directory consists of Aircrafts screen and Aircraft Detail screen. Besides, Aircraft Operation Bloc and Aircraft Form Bloc are placed for serving data in a form.

Common directory consists of shareable elements in the user interface. For example, they are dialogs, tile dividers, navigation bars and progress indicators.

Dashboard directory consists of Location Bloc, Map Bloc and the nested Blocs described in the Software Architecture section 4.1.2. Also, it contains all graphic elements that are able to see in the Dashboard screen, including the Search screen with the managing Blocs. The Dashboard is the most complex part of this application. It assembles data about zones, drones, and places and placing them into the map.

Devices directory consists of Devices screen and Device Detail screen. Besides, there is placed Device Operation Bloc, Device Detail Bloc, for control the switching elements in Device Detail Screen and Device Form Bloc for serving data in a form.

Flights directory consists of Flights screen, Flight Detail screen, In-flight screen and Plan a flight screens. Also, Flight Operation Bloc and Flight Form Bloc are placed for serving data in a form.

Intro directory consists of Login and Registration subdirectories. Both of them contain UI components and their logic. It is arranged for self-organized directories because of encapsulation.

Profile directory consists of the Profile screen and Profile Detail screen. Besides, there is placed Profile User Bloc and Profile Detail Form Bloc for serving data in a form.

4.2.4 Util

This part contains utilities that facilitate work in the codebase. It is separated into:

- **Error directory**,
- **Extensions directory**,
- **Preferences directory**,
- **Bloc Delegate class** – it extends generic Bloc Delegate, and it logs all actions on every Bloc and Hydrated Bloc,
- **Network Info class** – it checks internet connection (it uses DataConnectionChecker class),
- **Validator class** – it defines validation rules for e-mail address, password and full name.

4.2.4.1 Error

This directory contains classes:

- **Error_handling** – it contains a method that maps an error to action,
- **Server Exception** – it extends the general exception,

- **Server Failure** – it represents a returned fail message with the HTTP status code.

4.2.4.2 Extensions

The Extension is a feature that can add functionality without the base class changes. Besides, it helps the codebase facilitation and Extension can use even in cases where it is needed. The Extensions directory contains:

- Polygon Extension,
- Circle Extension,
- Number Extension,
- String Extension.

Polygon Extension offers to find out if a point consisted of latitude and longitude is lying in the given Polygon [54]. It is based on the mathematical problem when it is being looked for the result if y-axis of the given point goes through an even number of borderlines of the given Polygon. To solve how to find out if the y-axis goes through is based on the problem which deals with two lines go through each other. All the problem is described in this paper [55].

Circle Extension offers to find out if a point consisted of latitude and longitude is lying in the given Circle [56]. It is based on the *Pythagorean theorem* [57] that solves a computation of the longest line in a right triangle consisting of the spoken line and x and y-axis lines.

Viewport Extension contains *isOutOf(Viewport viewport)* method, that extends the base interface of LatLngBounds class [58] for the comparison if this viewport is out of the bounds of the given viewport. The implementation is simple. It compares all coordinates of these viewports each other. If this viewport is out of the bounds of the given, it returns true. Otherwise, it returns false.

String Extension contains methods for simplified work with Strings. These methods are:

- **isValidEmail()** – it checks if a given e-mail meets set Regular Expression [59],
- **isValidPassword()** – it checks if a given password meets set Regular Expression,
- **isBlank()** – it checks if a given string is empty or equal to an empty string,
- **isValidFullName()** – it checks if a given full name meets set Regular Expression.

Number Extension contains methods for the precision setting. The methods are:

- **halfSize()** – returns half of this number,
- **toLatLonPrecision()** – returns string of this number with precision on 6 decimal digits,
- **toAltitudePrecision()** – returns string of this number with precision on 4 decimal digits,
- **addLatitudeDistance(double distance)** – adds a given distance to this in latitude axis,
- **addLongitudeDistance(double distance, double latitude)** – adds a given distance to this in longitude axis.

Date Extension extends the base interface of the Date class and contains the get method *suffix*, which returns a suffix in English for the number of the day of the given date.

Translation Extension extends the base interface of the BuildContext class and contains *translate(String key)* method that mediates call of *Flutteri18n.translate(key)* method with key parameter. It returns a translation by phone setting properties, where the application is running, by the key in **en.json** and **cs.json** file in **assets/i18n** directory.

4.2.4.3 Preferences

This directory contains AppPreferences class that encapsulates Box class from Hive library, which provides NoSQL persistent data storage. The Hive library has a simple interface for work with the nested Box class. Further, it allows using Hive Annotations and define the objects for storing via an entity. AppPreferences class contains public methods:

- **saveTokens(String apiToken, String refreshToken)** – it calls _putToken(String token) and _putRefreshToken(String token) methods,
- **destroyTokens()** – it calls _deleteToken() and _deleteRefreshToken() methods,
- **getToken()** – it gets a value by API_TOKEN constant,
- **getRefreshToken()** – it gets a value by API_REFRESH_TOKEN constant,
- **isUserLoggedIn()** – it checks if getToken() and getRefreshToken() return not empty string (If so, user is still logged in, otherwise logged out.),

- **getRecentSearchResult()** – it returns stored user search recent results,
- **updateRecentSearchResult(SearchObject currentSearchedResult)** – it updates stored user search recent results.

AppPreferences class contains private methods:

- **_put(String key, value)** – it puts a value by the given key into Box object storage,
- **_putToken(String token)** – it calls _put(String key, value) where the key is API_TOKEN constant,
- **_putRefreshToken(String token)** – it calls _put(String key, value) where the key is API_REFRESH_TOKEN constant,
- **_deleteToken()** – it deletes a value by API_TOKEN constant,
- **_deleteRefreshToken()** – it deletes a value by API_REFRESH_TOKEN constant.

Chapter 5

User Interface Design

This chapter is focused on the user interface design of the Dronetag mobile application. It starts with an explanation of difference between UI (User Interface) and UX (User Experience), and continues with a description about the user interface structure of the application. So, it contains a list of screens that are shown to users when they go through the application.

The main difference between UI and UX is in the approach. UI approach emphasizes a combination of visual elements such that the final concept of visualization fits together. UX approach emphasizes an arrangement of these visual elements such that their arrangement was logical, and it was close to human thinking. [60] There are many visual elements that can interact with users. They can be:

- A text label,
- A text field,
- A button,
- A checkbox,
- A List view,
- A menu,
- An icon,
- An image,
- And others.

UI approach deals with their skin - for example, a color, size, shadow, border, shape, and others. UX approach focuses on the size and position on the screen. The arrangement of elements in screens should be comprehensive and should make sense to users.

The user interface design phase usually includes Lo-Fi (Low Fidelity) and Hi-Fi (High Fidelity) prototyping. [61] Lo-Fi, how the name hints, is a simplified sketch without colors with a purpose to incorporate the elements on a screen together. It is cheaper than Hi-Fi and takes only a few hours to finish. The benefits are:

- Focus on design and concepts,
- Focus on design and concepts,
- Accessible to everyone. [62]

In opposite, Hi-Fi, how the name hints, includes a fidelity design that should correspond to the final product. It is more expensive than Lo-Fi and usually takes a few days. The benefits are:

- More familiar to users,
- Pinpoint specific components to test,
- More presentable to stakeholders. [62]

To more information about fidelity, anyone can read this article [62].

Lo-Fi prototype has not been created because Dronetag is Start-up, a company with a limited budget, so there is only the Hi-Fi prototype. When Hi-Fi was being made, it was inspired by a few competitor applications, which show permitted flight zones and danger areas. These applications are described in the Related projects chapter 1, so they will not be discussed anymore.

5.1 Hi-Fi Prototype

This section consists of screens in the application with a detailed description of their elements. The Hi-Fi prototype was made by Marian Hlaváč in Adobe XD [63] and is available online [64]. It was emphasized to the simplicity and briefness and kept the focus on the essential UI and UX design rules. These rules are based on "Jakob Nielsen's 10 general principles for interaction design". [65] It describes key aspects of User Interface Design. The heuristics by Jakob Nielsen are the following:

1. Visibility of system status,
2. Match between system and the real world,
3. User control and freedom,
4. Consistency and standards,
5. Error prevention,

6. Recognition rather than recall,
7. Flexibility and efficiency of use,
8. Aesthetic and minimalist design,
9. Help users recognize, diagnose, and recover from errors,
10. Help and documentation.

Applications built on these heuristics will be successful and useful because these heuristics are based on various psychological researches and usability testing with ordinary users.

5.1.1 Dashboard

The Dashboard screen is immediately after the Splash screen. It is counted on the fact users will spend most of the time on the Dashboard screen, and so it contains the main functionalities. This screen is divided into Top panel and Map controls. These elements are in the higher stack layer above the map where are placed flying drones, place pins and restricted areas. Besides, it allows displaying without wasteful POIs (Point of Interest) [66], satellite map or standard map, which contains POIs, precisely like Google Maps application.

The Top panel is consists of:

- **Device status** – it contains current information about default device,
- **Search button** – it opens the Search screen for searching of places, devices, aircrafts and zones,
- **Profile button** – it open the Profile screen that represents the main menu of the whole application.

If a user has already logged in and planned a flight, Profile button contains a light blue circle with the number of planned flights in the top right on the circuit of that circle button. It is able to see on the picture 5.1.

The Map controls are consists of:

- **My location button** – it redirects to his position due to his location by GPS coordinates,
- **Map layers button** – it is an offer allowing users to choose a map style,
- **Fly now button** – this button shows drop down menu with Fly now and Plan a flight options for already logged in users, and Log in and sign up button for unauthorized users.

These are additional elements that are shown on the Dashboard screen:

- Drone detail panel, (Figure 5.2)
- Place detail panel,
- Zone detail panel, (Figure 5.3)
- Zone selection panel,
- Map layer panel.

The Drone detail panel is for showing necessary information and a flying drone and its flight parameters. Besides, it contains the button to redirect to the full drone detail screen, where users can see all information about live flight immediately.

The Place detail panel contains necessary information about the place. Also, if users want to plan a flight from this place, it allows them to pin this place on the map until they unpins it and Plan a flight button. Place pinning into the map is demonstrated by a color change of the pin on the map and icon in the panel.

The Zone detail panel contains necessary information about the given zone, such as a lower altitude level, upper altitude level, name of the zone, and validation from and zone status.

The Zone selection is used to select a zone if a user clicks to a place in the map where is an intersection of more zones. Simultaneously, when this selection is shown, the selected zones are highlighted. If a user chooses a zone, the zone will show Zone detail panel and it will be highlighted only the particular zone instead of the intersection.

It was focused on the primary purpose of informing users about drones and restricted areas around them. So, it was decided to keep a minimalistic design because the users needs to emphasize to important elements to them.

5.1.2 Login and Registration Screens

The Login screen contains buttons for logging in and signing up to the system. Login buttons allow logging in via e-mail, Google account, and it is going to add logging in via Apple account. Sign up button allows signing in only via e-mail.

The Log in Screen consists of two text fields. The first is for typing an e-mail and the other is for typing a password. After success logging in, the users are redirected to the Dashboard screen.

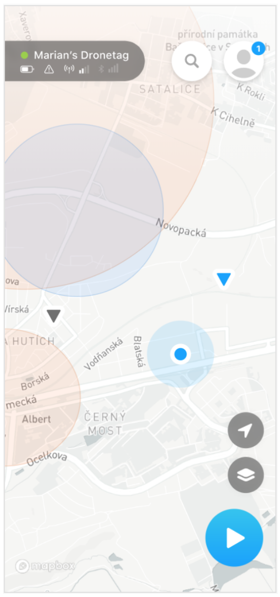
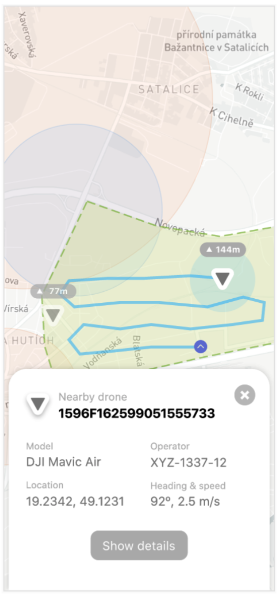


Figure 5.1: Dashboard



Nearby drone
1596F162599051555733

Model
DJI Mavic Air

Operator
XYZ-1337-12

Location
19.2342, 49.1231

Heading & speed
92°, 2.5 m/s

Figure 5.2: Drone detail

The Sign up screen is consist of the following three text fields:

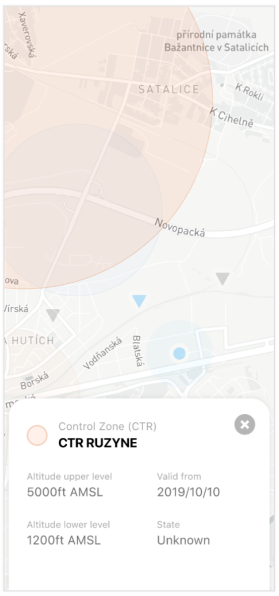
- The first is for typing an e-mail address through the user will log in to the system,
- The second is for typing a password,
- The third is for typing an optional full name.

The users have to agree with terms of using before they finish the registration. After successful signing up, the users are logged in and redirected to the Dashboard screen.

5.1.3 Profile Screens

The Profile screen (Figure 5.4) represents the main menu of the application. Users have granted access to their profile detail, My management container, and the users can see the set default aircraft and device. Besides, if they belong to an organization, it shows them the organization name and the button to switch the fleet mode. My management container contains My Flights, My Devices and My Aircrafts buttons.

The Profile detail screen contains the user properties like the full name, phone number and country. Also, it allows changing the user's contact e-mail and password.



Control Zone (CTR)
CTR RUZYNE

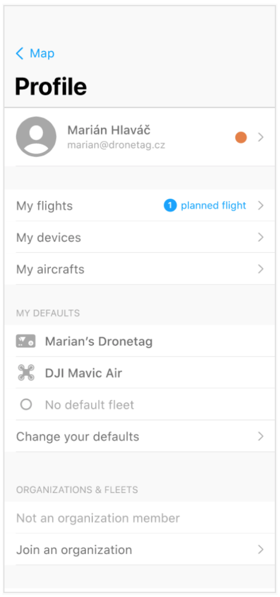
Altitude upper level
5000ft AMSL

Valid from
2019/10/10

Altitude lower level
1200ft AMSL

State
Unknown

Figure 5.3: Zone detail



Map

Profile

Marián Hlaváč
marian@dronetag.cz

My flights — 1 planned flight
My devices
My aircrafts

MY DEFAULTS
Marian's Dronetag
DJI Mavic Air
No default fleet
Change your defaults

ORGANIZATIONS & FLEETS
Not an organization member
Join an organization

Figure 5.4: Profile

5.1.4 Device Screens

The Devices screen (Figure 5.5) will be showed after users click on My Devices button in the Profile screen. This screen contains a list of registered Dronetag devices and Add button. The Add button redirects the users to the Device registration screen.

The Device screen contains all information about the given device, including the real–time data. Users can set the device as default or set the device preferences by their own needs.

The Device registration screen is for registration of a new device that the users bought. It contains two text fields for typing - a serial number, and an optional name for better identification. After successful registration, the users are redirected to the Devices screen.

5.1.5 Aircraft Screens

The Aircrafts screen contains a list of added aircrafts. Besides, users can add a new drone via the Add button. This button redirects users to the Aircraft screen with blank text fields and selection elements that are set to initial value. Users will get to this screen from Profile screen after the click to the My Aircrafts button.

The Aircraft screen is to display the detail of an aircraft. Simultaneously, it is used for adding new aircraft. A user will get to it after choose a one from the list in Aircrafts screen. This screen contains a form with three text fields:

- The first is for type name,
- The second is for type UAS (Unmanned aircraft systems) [23] Operator ID what identify a person that has permission to flight with drones and passed the pilot exams,
- The third is for setting the weight of aircraft.

Also, this screen contains two selection elements for choosing the vendor and its model.

5.1.6 Flight Screens

The Flights screen (Figure 5.6) is for displaying a list of the flights. This screen contains flight statistics and allows finding certain flights by chosen filters. By every flight, it shows its state if it is planned and current flight, planned start and finish dates, and a map with the flight telemetry. Besides, it contains the button for exporting the whole flight history. If users choose a planned flight, it shows the Flight plan screen, which is a planned flight summary and consists of various parameters. It is possible to show the Flight detail screen for finished flights, and the In-flight detail screen for current flights.

The Flight detail screen contains all information about a flight, including the option to replay flight trajectory from start to the end. The In-flight detail screen contains information about the flight in real-time. Thanks to the Sliding up panel the user is able to see more information and still check if a drone did not escape the reservation area.

The Plan a flight is a group of three screens that represent a wizard for flight plan. In the first screen, the users set identification properties and planned time of the flight. In the second screen, the users set a polygon or circle for the planned flight and its maximum altitude. In the last screen, the users confirm set parameters of the planned flight - it is like a summary.

5.1.7 Search Screen

The Opened Search screen is the first what users see when they click on the Search button in the Dashboard. It shows recent results that the users have searched in the last time. The result can be a drone, aircraft, place or zone.

If users use the Search text field and types a text string, it will show them a result with an icon for better identification. If users choose an item in the list of results, it will redirect them to any of the Detail screens. The screen can be either the Aircraft detail screen, the Device detail screen, or a place

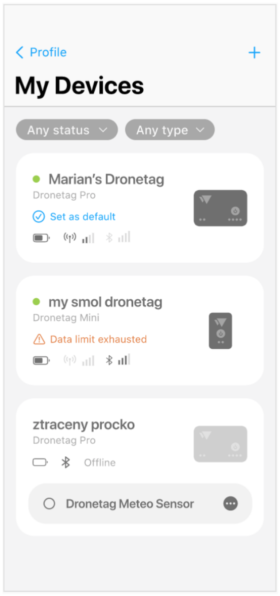


Figure 5.5: Devices

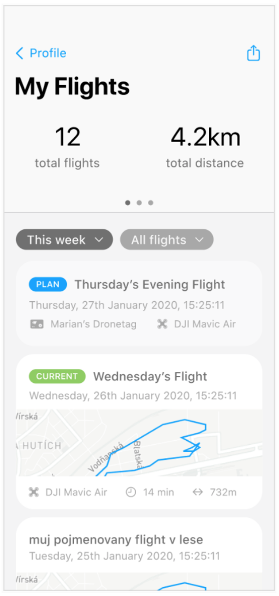

Profile
My Flights
12
total flights
4.2km
total distance
This week
All flights
PLAN
Thursday's Evening Flight
Thursday, 27th January 2020, 15:25:11
Marian's Dronetag
DJI Mavic Air
CURRENT
Wednesday's Flight
Wednesday, 26th January 2020, 15:25:11
DJI Mavic Air
14 min
732m
muj pojmenovany flight v lese
Tuesday, 25th January 2020, 15:25:11

Figure 5.6: Flights

on the map in the Dashboard. After choosing that item, it will appear in the list of the Recent results.

5.2 Usability Testing

This section talks about usability testing what is a phase in the user interface design cycle. This phase involves testing with users who get instruction by a scenario, and the task that they get they should do in their natural way. The primary purpose of this testing is the detection of harmful design elements, and the determination of the arrangement mistakes. The elements can be arranged in a screen, part of a screen, menu or drop-down menu. It depends on the purpose of the concrete application.

At first, the Hi-Fi prototype was designed and inspired by the competitor applications. In the design was effort to learn from design mistakes of the competitor applications and solve their flaws. It had to be decided if it is better to place a sliding panel with advisories on Dashboard or an animated button for a flight planning, how should look the profile screen and its icon, or if it is useful to pin a place into the map. After that, it has been organized the usability testing with ordinary people who could be Dronetag's potential customers.

During the testing, it was detected a few crucial mistakes in the design. They are accurately described in the following part of this section.

5.2.1 Results

This part contains feedback from usability testing. There has been organized two usability tests. The first was at the beginning of mobile application development before the implementing of the essential application functions was started. The other was before the release of the alpha version of this application. The following paragraphs relate to the usability testing before the alpha version.

It has been tested with six users. One of them was from the Faculty of transportation sciences of Czech Technical University in Prague. The others were colleagues from the Faculty of information technology of the same university. Many of them were bachelor students, so it means they are between 20 and 25 of age. The student of the Faculty of transportation sciences gave an insight into what would expect users with flight knowledge. He describes how the Dashboard map should behave and how to understand various map layers and flight levels. My Faculty's colleagues were in the role of typical users and helped to understand what they expect of the used elements in the screens. For the briefness, there were chosen and cited three of them. Due to GDPR, their names were anonymized.

Person One:

- ”I would like to see separated groups in the Search screen with Headline text to distinguish what the searched thing is.”
- ”I like the Full drone detail screen. It contains everything that I want to know. The map is impressive.”

Person Two:

- ”It would be nice to have a list of pinned places on the Dashboard map. If I pin many places, I will get lost between them.”
- ”I like the Country list selection with the Search bar in the Profile screen. It is good-looking and useful.”
- ”The application is stable in the Alpha version. I like the simplicity of the Dashboard screen.”

Person Three:

- ”The place pin in the Dashboard map is too small, and I cannot see it.”
- ”The zone colors with opacity are too light for sunlight. I would not have to see them outside.”
- ”I like the tiles in My Aircrafts and My Devices screens. The swiping for delete looks excellent. However, I am missing an option to select these objects and their actions (For example, delete, share, or change order).”

5.2.2 Main Mistakes in User Interface Design

During the usability testing, there were discovered the critical findings that are described in this part. Some of them were expected. Some of them were not expected and forced to change the future goals in the user interface design.

For example, the following flaws were found:

1. It is not so intuitive to expect the searching drones and devices from the Search button in the Dashboard. The users expect only searching places, whereas it is a part of the Dashboard, where the map is the essential element.

2. It is not obvious that the bottom panel is possible to pull up. So, it is the first of the reasons to decide to omit the bottom sliding up panel with advisories. The other is that it is not known and approved the final legislation so far, and it is still in progress.

3. The assigning of the drone to the device is not useful. It is better to allow users to set a default device and default drone, and allow the users to change them when they are planning a flight.

4. Users appreciate an option to move with the map in the In-flight screen.

This knowledge has had a significant impact on changing screens to reach a better attention of users.

Chapter 6

Deployment and Testing

This chapter describes the required steps for setting the development environment and describes the configuration files. Besides, it describes automation testing methods, and a summary of verified functions in the application by these testing methods.

6.1 Build Instructions

The build is divided into main 4 steps:

1. downloading dependencies via **flutter pub get** command,
2. launching generating of the classes that creates themselves by the annotation settings via **flutter pub run build_runner** command
3. launching generating APK file for Android via **flutter build apk --release** command and
4. launching generating APP file for iOS in ios/Runner bundle.

The last two steps can be run concurrently because they are independent of each other. All this build process allows to be fully autonomous and connected to CI (Continuous Integration) and CD (Continuous Delivery). Thanks to this fact, the development speed up the whole software process, and it does not need any information technology operators. This kind of development is called DevOps (abbreviation of development and information technology operations), and now its popularity is increasing because of saving time and money.

6.2 Configuration Files

For using Continuous Integration and Continuous Delivery is needed to set the right configuration. Therefore, there is a need to create single configuration files for every environment. For this purpose, we used the environment_config library that allows easy management and provides the generator. This generator allows to use these settings in the codebase by the given parameters. The configuration YAML [67] file keeps the setting of these variables, where we choose the required configuration, and the generator makes available it via an interface in the codebase. There is needed to notice that the cross-platform frameworks like Flutter allow to set the separate configuration for every platform on iOS and Android. Unfortunately, there are even cases when the setting applies as similar as on both of the platforms. For example, it can be a URL (Uniform Resource Locator) addresses, client ID, client secrets, and a path to a log file. This change will be performed in the specific bundle if it is needed to distinguish a configuration for each of the platforms alone. A typical example could be differentiated setting of Google Maps widget, where it is needed to set activated Google Maps API key. Although these platforms have different architecture, so its configuration is different.

Immediately at the beginning of the development phase, when new members join the team, they must clone **mobile-app** folder from the git repository on BitBucket server. After that, they must add files **auth_config.json** and **google_map_api_key.json** in directory **assets/config**.

auth_config.json file in the format:

```
{
    "clientId": "...",
    "clientSecret": "..."
}
```

This file is used for the security of the client access and in case it would be stolen from one environment, it will not be applicable in another environment.

google_map_api_key.json file is in the format:

```
{
    "apiKey": "..."
}
```

This file is for access to the information, which are provided by Google Cloud Platform API. In our project, we use Google Maps API for downloading map styles, data about a searching place, and the Autocomplete function that suggests accurate and close results by the given substring. All mentioned files meet the JSON format standard specification.

6.2.1 Develop Environment

The configuration files in this project contain setting for the local running platform launched as the Kubernetes cluster. The part of the cluster is a docker container with the backend implementation. More information about the platform is available in the Dronetag web infrastructure chapter 2.

6.2.2 Test Environment (Staging)

The test environment is called Staging and is used for testing and debugging new functionalities, features, and software quality assurance.[68] In this environment, is possible to simulate the arbitrary model situation and check how the application behaves in these edge case situations and if it is stable in each scenario. Also, it is possible to verify the fact if a fault occurs, that the clients can adequately respond to this situation and catch thrown exceptions.

6.2.3 Production Environment

The production environment is used for real application operations, and it should contain no errors created during the development phase. This environment is crucial for the Dronetag company business, and the customers who use the product must have a feeling about the flawless application whenever they use it. Configuration files in this environment contain the connection settings to the production server, where is being run the cluster with the backend instance.

6.3 Code Testing

The testing has in the software development the critical usage if the development runs according to an agile methodology. It means that it is about small regular deliveries of software pieces that include certain functionality or bug fixing. These interferences in the already deployed application are needed to test its correct functionality to prevent potential problems in the future. The biggest problem occurs where dependencies create. So, if a functionality breaks down at the beginning, the functionalities depending on that root functionality will return bad results or causing instability of the application. The advantage is that the tests can be repeatedly run, and if it is needed, they can be changed easily. So, the testing is a phase in the software development that allows the automation in the CI and CD.

Besides, this chapter describes types of tests and functionalities in the application, which are covered by these tests. It includes a description of how the given functions should behave in certain situations.

6.3.1 Unit Testing

This section emphasizes to unit tests. In Flutter, unit tests run with **flutter test** command from the console prompt. Thanks to this, it is easy to incorporate them into CI and CD phase, where the test can run automatically. There are libraries used for unit testing in Flutter:

- test,
- flutter_test,
- Mockito,
- Bloc_test,
- And others.

It was emphasized to functionalities that ensure to get data from remote sources. The correctness of these functionalities is necessary because the application must be stable even when the backend is out-of-service. Thus, these classes are tested to the three essential situations:

- **Success** - when the data are successfully received,
- **Failure** - when a failure occurs during loading,
- **Disconnected** - when a phone is in an out-of-service area.

So, Unit tests in the application cover the following repositories that are serving data from the remote sources:

- User Repository,
- Device Repository,
- Aircraft Repository,
- Flight Repository,
- Zone Repository,
- Search Repository.

6.3.2 Integration Testing

This section clarifies what Integration tests are. These tests verify whether the individual parts tested by unit tests can cooperate and work well as a whole. For example, if an application has a multi-layer architecture, it is needed to verify if these layers passing data through by the expectation. So, an integration test consists of smaller parts tested by unit tests. The ntegration tests have not been implemented in the mobile application because there was no time to deal with this type of the tests.

6.3.3 Mocker

Mocker is used for flight drone simulations. Mocker is written in Python [69] and sends messages to backend endpoint for the testing purpose. It expects a serial number parameter and allows to specify an optional take-off position parameter. It allows mocking more drones concurrently. The advantage of this Mocker is that it is not needed to have a flying physical drone with an attached Dronetag device in the air during the development. It is useful to run this Mocker for a simulated behavior because there is not needed anything else.

Conclusion

In this thesis, the problem of lawbreaking by flight drones was discussed. Also, zone types which the pilots have to avoid were described, as well as the flight levels. Besides, there were clarified the differences among these zone types. An analysis of related solutions for drone pilot support was conducted, and the key differences among them were described in the Related projects chapter 1. There was introduced the technology stack which provides data, data modelling, and connectivity for these technologies in the Backend infrastructure chapter 2. The basics of Flutter and the reasons why it was chosen for cross-platform mobile application development were discussed, including a brief comparison of the competing cross-platform frameworks. In addition, the Bloc pattern was clarified and compared with its extended class, Hydrated Bloc.

There was described the difference between software architecture and design. Then, it was emphasized the designed code structure of mobile application implementation that was divided into the software architecture and design components, according to the concept appproach. Further, it was found that the UX/UI design initially had some flaws, which was revealed during the usability testing. Hence, it is still improving and changing, based on the results from the usability testing of the first version of the mobile application. There was described how to realize the implementation of this thesis, including the necessary steps to run the project in the development environment. The implementation meets all functional requirements, specifically drawing flight zones on the map, showing real-time data about flying drones with a Dronetag device attached, and flight planning and management of aircrafts, devices and flight history. The environment description also contains the configuration files which should not be versioned with the version systems for security reasons. It was also discussed software quality assurance and which methods are used in the mobile application. There were clarified the advantages and use cases of automated tests and chosen functions tested by Unit tests.

There was mentioned that one of the Dronetag products is a mobile application that will ensure safer drone operations; this application is used as a client. Thus, it is directly dependent on the entire platform, especially on the Backend and Live Service. Hence, the development was divided into two main phases, called Alpha and Beta.

For the Alpha release was set the date to May 31st, 2020. This version contains essential functionality like watching zones, drones, devices, and flight management, including searching and flight planning in two modes. The first mode is ”Fly now,” which allows users to record his flight quickly. The other is ”Plan a flight,” which allows users to plan a flight for a specific time and place. As such, the Alpha version will provide the main application functionality.

Over the coming months, there will being prepared the second version called Beta. This version will emphasize performance optimization and use some data in an offline mode as there is required to ensure full application functionality even in places without service. In addition, it includes the complete functionality realization of the fleet management for organizations. This functionality would be suitable for industrial purposes and ensure the safety of a larger group of drones in the reserved area. Most of the customers would be companies delivering goods, monitoring airports, and transferring medical materials to inaccessible places. Overall, the Beta version will focus on the optimization and improvement of any problems that have arisen in the Alpha version.

In conclusion, it is important to emphasize the fact that the mobile application structure is still growing. This fact means there are still to be implemented new functions that will contribute to more widespread use of the application. In addition, there will be a need to stay up-to-date with new developments in the use of drones in business, always improving and adapting these technologies to the customers’ requirements. The Dronetag company is reactive to these requirements, so it offers a product that satisfies them and can be used worldwide.

Bibliography

[1] flutter-dev. *Technical overview - Flutter [online]*. [Accessed: 2020-05-16]. Available from: `https://flutter.dev/docs/resources/technical-overview`

[2] (felangel), F. A. *Bloc, a Dart package [online]*. [Accessed: 2020-04-26]. Available from: `https://github.com/felangel/bloc/blob/master/packages/bloc/README.md`

[3] D'Onfro, J. Amazon's New Delivery Drone Will Start Shipping Packages 'In A Matter Of Months'. *Forbes Media LLC*, June 2019, [Accessed: 2020-05-09]. Available from: `https://www.forbes.com/sites/jilliandonfro/2019/06/05/amazon-new-delivery-drone-remars-warehouse-robots-alexa-prediction/`

[4] of the Czech Republic, A. N. S. Airspace of the Czech republic. online, [Accessed: 2020-05-21]. Available from: `https://aim.rlp.cz/vfrmanual/actual/enr_1_en.html`

[5] Investment; CzechInvest, B. D. A. Results from the Space Application Hackathon - CSW 2020. online, [Accessed: 2020-05-27]. Available from: `http://www.czechspaceyear.com/results-from-space-application-hackathon/`

[6] AIRMAP, I. AirMap websites. online, [Accessed: 2020-04-08]. Available from: `https://www.airmap.com`

[7] Mapbox. Mapbox. online, [Accessed: 2020-05-22]. Available from: `https://www.mapbox.com`

[8] for the Safety of Air Navigation, T. E. O. EUROCONTROL — Supporting European Aviation — EUROCONTROL. online, [Accessed: 2020-05-22]. Available from: `https://www.eurocontrol.int`

[9] AirMap Inc. *AirMap Developers [online]*. [Accessed: 2020-05-22]. Available from: https://developers.airmap.com

[10] Angel, A. GuardianUTM. online, [Accessed: 2020-05-22]. Available from: https://docs.altitudeangel.com/docs

[11] Services, N. A. T. Safety Apps - Dronesafe. online, [Accessed: 2020-05-25]. Available from: https://dronesafe.uk/safety-apps/

[12] s.r.o., U. MAIA - Mobile Aircraft Identification Application — MAIA. online, [Accessed: 2020-05-22]. Available from: https://flymaia.com

[13] Kittyhawk.io, I. Kittyhawk - Safety Apps - Drone Fleet Management System. online, [Accessed: 2020-05-25]. Available from: https://kittyhawk.io

[14] Institute, C. H. CHMI Portal : Home. online, [Accessed: 2020-05-26]. Available from: http://portal.chmi.cz/?l=en

[15] Organization, I. C. A. Home — International Civil Aviation Organization. online, [Accessed: 2020-05-26]. Available from: https://www.icao.int/Pages/default.aspx

[16] Air Navigation Services of the Czech Republic. *Létejte zodpovědně [online]*. [Accessed: 2020-05-21]. Available from: https://letejtezodpovedne.cz

[17] Red Hat, Inc. *What is Docker? — Opensource.com [online]*. [Accessed: 2020-05-10]. Available from: https://opensource.com/resources/what-docker

[18] Docker Inc. *What is a Container? — App Containerization — Docker [online]*. [Accessed: 2020-05-10]. Available from: https://www.docker.com/resources/what-container

[19] Docker Inc. *Overview of Docker Compose — Docker Documentation [online]*. [Accessed: 2020-05-10]. Available from: https://docs.docker.com/compose/

[20] The Kubernetes Authors. *Production-Grade Container Orchestration - Kubernetes [online]*. [Accessed: 2020-05-10]. Available from: https://kubernetes.io

[21] Inc., G. Borg, Omega, and Kubernetes. *acmqueue*, March 2016, [Accessed: 2020-05-17]. Available from: https://queue.acm.org/detail.cfm?id=2898444

[22] Dronetag s.r.o. *Datový model webové platformy :: Dokumentační web Dronetag [online]*. [Accessed: 2020-04-19]. Available from: `https://docs.dronetag.cz/kbase/platform/datamodel.html`

[23] EUROCONTROL. Unmanned aircraft systems — EUROCONTROL. online, [Accessed: 2020-05-27]. Available from: `https://www.eurocontrol.int/unmanned-aircraft-systems`

[24] The Internet Engineering Task Force (IETF). *GeoJSON [online]*. [Accessed: 2020-05-17]. Available from: `https://geojson.org`

[25] Redis labs. *Redis [online]*. [Accessed: 2020-05-09]. Available from: `https://redis.io`

[26] Jagtap, S. Flutter vs React Native: A Developer's Perspective. *nevercode.io Blog*, April 2020, [Accessed: 2020-05-16]. Available from: `https://nevercode.io/blog/flutter-vs-react-native-a-developers-perspective/`

[27] Sharma, N. React Native vs. Xamarin vs. Ionic vs. Flutter: Which is better? *Apptunix*, September 2018, [Accessed: 2020-05-21]. Available from: `https://www.apptunix.com/blog/frameworks-cross-platform-mobile-app-development/`

[28] Dart Community. *The Dart type system — Dart [online]*. [Accessed: 2020-05-16]. Available from: `https://dart.dev/guides/language/sound-dart`

[29] Nader, Y. React Native vs Flutter. *hackr.io Blog*, April 2020, [Accessed: 2020-05-16]. Available from: `https://hackr.io/blog/react-native-vs-flutter`

[30] Alessandria, S. *Flutter Projects: A practical, project-based guide to building real-world cross-platform mobile applications and games.* Packt Publishing Ltd, first edition, ISBN 9781838642532.

[31] Google, Inc. *RenderObjectWidget class [online]*. [Accessed: 2020-04-27]. Available from: `https://api.flutter.dev/flutter/widgets/RenderObjectWidget-class.html`

[32] Google, Inc. *RenderObject class [online]*. [Accessed: 2020-04-27]. Available from: `https://api.flutter.dev/flutter/rendering/RenderObject-class.html`

[33] Google, Inc. *StatelessWidget class [online]*. [Accessed: 2020-04-08]. Available from: `https://api.flutter.dev/flutter/widgets/StatelessWidget-class.html`

[34] Google, Inc. *State class [online]*. [Accessed: 2020-04-27]. Available from: `https://api.flutter.dev/flutter/widgets/State-class.html`

[35] Google, Inc. *setState method [online]*. [Accessed: 2020-04-27]. Available from: `https://api.flutter.dev/flutter/widgets/State/setState.html`

[36] Google, Inc. *StatefulWidget class [online]*. [Accessed: 2020-04-08]. Available from: `https://api.flutter.dev/flutter/widgets/StatefulWidget-class.html`

[37] Google, I. *BuildContext class, dependOnInheritedWidgetOfExactType¡T extends InheritedWidget¿ method [online]*. [Accessed: 2020-04-26]. Available from: `https://api.flutter.dev/flutter/widgets/BuildContext/dependOnInheritedWidgetOfExactType.html`

[38] Google, Inc. *InheritedWidget class [online]*. [Accessed: 2020-04-08]. Available from: `https://api.flutter.dev/flutter/widgets/InheritedWidget-class.html`

[39] Boelens, D. Reactive Programming - Streams - BLoC. *Didier Boelens Blog*, August 2018, [Accessed: 2020-05-10]. Available from: `https://www.didierboelens.com/2018/08/reactive-programming-streams-bloc/`

[40] (felangel), F. A. *Hydrated Bloc, a Dart package [online]*. [Accessed: 2020-05-16]. Available from: `https://pub.dev/packages/hydrated_bloc`

[41] Community, F. *Path Provider, a Dart package [online]*. [Accessed: 2020-05-16]. Available from: `https://pub.dev/packages/path_provider`

[42] (felangel), F. A. *Example - Hydrated Bloc, a Dart package [online]*. [Accessed: 2020-05-16]. Available from: `https://github.com/felangel/hydrated_bloc/tree/master/example`

[43] Angelov, F. Caching Bloc State with “Hydrated Bloc” - Flutter Community. *Medium*, June 2019, [Accessed: 2020-05-16]. Available from: `https://medium.com/flutter-community/caching-bloc-state-with-hydrated-bloc-e565f81520a4`

[44] (felangel), F. A. *Hydrated Bloc Storage, a Dart package [online]*. [Accessed: 2020-05-16]. Available from: `https://github.com/felangel/hydrated_bloc/blob/master/lib/src/hydrated_bloc_storage.dart`

[45] Google Inc. *Cupertino (iOS-style) widgets - Flutter [online]*. [Accessed: 2020-05-20]. Available from: `https://flutter.dev/docs/development/ui/widgets/cupertino`

[46] Bohn, D. Google's software design is having a reformation. *The Verge*, May 2018, [Accessed: 2020-05-20]. Available from: `https://www.theverge.com/2018/5/10/17339230/google-material-design-theme-update-new-tools-matias-duarte`

[47] Google Inc. *Components - Material Design [online]*. [Accessed: 2020-05-20]. Available from: `https://material.io/components`

[48] Muhammad Ali Babar, I. M., Alan W. Brown. *Agile Software Architecture*. Elsevier Science and Technology, first edition, ISBN 9780124078857.

[49] Robinson, J. A. *Software Design for Engineers and Scientists*. Elsevier Science and Technology, first edition, ISBN 9780080474403.

[50] Edward Curry, P. G. Flexible Self-Management Using the Model-View-Controller Pattern. *IEEE Software*, April 2008, [Accessed: 2020-05-25]. Available from: `https://ieeexplore.ieee.org/document/4497770`

[51] Dart Community. *Dart packages [online]*. [Accessed: 2020-05-10]. Available from: `https://pub.dev`

[52] Fowler, M. Inversion of Control Containers and the Dependency Injection pattern. *martinfowler.com*, January 2004, [Accessed: 2020-05-18]. Available from: `https://martinfowler.com/articles/injection.html`

[53] Auth0. *JSON Web Tokens - jwt.io [online]*. [Accessed: 2020-05-18]. Available from: `https://jwt.io`

[54] Google Inc. *Simple Polygon — Maps JavaScript API — Google Developers [online]*. [Accessed: 2020-05-21]. Available from: `https://developers.google.com/maps/documentation/javascript/examples/polygon-simple`

[55] Prosser, P. Geometric Algorithms. *Glasgow University*, March 2000, [Accessed: 2020-05-21]. Available from: `http://www.dcs.gla.ac.uk/~pat/52233/slides/Geometry1x1.pdf`

[56] Google Inc. *Circles — Maps JavaScript API — Google Developers [online]*. [Accessed: 2020-05-21]. Available from: `https://developers.google.com/maps/documentation/javascript/examples/circle-simple`

[57] Hahn, R. *The Metaphysics of the Pythagorean Theorem*. SUNY Press, first edition, ISBN 9781438464893.

[58] Google Inc. *LatLngBounds — Google API for Android — Google Developers [online]*. [Accessed: 2020-05-22]. Available from: `https://developers.google.com/android/reference/com/google/android/gms/maps/model/LatLngBounds`

[59] Tourlakis, G. *Theory of Computation.* John Wiley and Sons, Incorporated, first edition, ISBN 9781118315330.

[60] Ben Coleman, D. G. *Designing UX: prototyping.* SitePoint, 2017, ISBN 9780994347084, [Accessed: 2020-05-25].

[61] Jonathan Arnowitz, N. B., Michael Arent. *Effective Prototyping for Software Makers.* Elsevier Science and Technology, first edition, ISBN 9780080468969.

[62] Esposito, E. Low-fidelity vs. high-fidelity prototyping. *Invision Design*, May 2018, [Accessed: 2020-05-17]. Available from: `https://www.invisionapp.com/inside-design/low-fi-vs-hi-fi-prototyping/`

[63] Inc., A. UI/UX design and collaboration tool — Adobe XD. online, [Accessed: 2020-05-25]. Available from: `https://www.adobe.com/products/xd.html`

[64] Inc., A. PH1-MVP. online, [Accessed: 2020-05-28]. Available from: `https://xd.adobe.com/view/c0ace277-443a-49d6-5eaa-7c3f3951fdc3-b69d/`

[65] Nielsen, J. 10 Usability Heuristics for User Interface Design. *nngroup.com*, April 1994, [Accessed: 2020-05-17]. Available from: `https://www.nngroup.com/articles/ten-usability-heuristics/`

[66] Google Inc. *Businesses and Other Points of Interest [online].* [Accessed: 2020-05-19]. Available from: `https://developers.google.com/maps/documentation/android-sdk/poi`

[67] The YAML Project. *The Offical YAML Web Site [online].* [Accessed: 2020-05-27]. Available from: `https://yaml.org`

[68] Chemuturi, M. *Mastering Software Quality Assurance.* J. Ross Publishing, first edition, ISBN 978-1-60427-032-7.

[69] Python Software Foundation. *Our Documentation — Python.org [online].* [Accessed: 2020-05-27]. Available from: `https://www.python.org/doc/`

Appendix **A**

Acronyms

AGL Above Ground Level. 2

AMSL Above Mean Sea Level. 1

API Application Programming Interface. 5, 10–12, 30, 32, 36, 38, 40, 56

APK Android Package. 55

BLoC Business Logic Component. 23, 24, 29, 32

CAA Civil Aviation Authority. 6

CD Continuous Delivery. 55, 57, 58

CHMI Czech Hydrometeorological Institute. 8

CI Continuous Integration. 55, 57, 58

CSS Cascade Style Sheet. 20

CTR Control zone. 3

dB Decibel. 34

DevOps Development and Operations. 55

DI Dependency Injection. 31

EAD European AIS Database. 4

GDPR General Data Protection Regulation. 53

GeoJSON Geographical JSON. 15, 17

GPS Global Positioning System. 34, 47

Hi-Fi High Fidelity. 46, 52

HTML HyperText Markup Language. 20

HTTP Hypertext Transfer Protocol. 31, 32, 37, 42

ICAO International Civil Aviation Organization. 8

IoC Inversion of Control. 31

IoT Internet of Things. 11

JSON Java Script Object Notation. 25, 31, 32, 56

JWT JSON Web Token. 32, 39

K8s Kubernetes. 13

LAANC Low Altitude Authorization and Notification Capability. 4

Lo-Fi Low Fidelity. 46

MVC Model-View-Controller. 29

MVP Model-View-Presenter. 29

MVP Most Value Product. 20

MVVM Model-View-ViewModel. 29

NATS National Air Traffic Services. 6

NoSQL Not only Structure Query Language. 30, 32, 43

NOTAM Notice To Airmen. 8

POI Point of Interest. 47

RSRP Reference Signal Receive Power. 14

SDK Software Development Kit. 5, 20, 30

SMS Short Message Service. 4

SVG Scalable Vector Graphics. 31

UAS Unmanned aircraft systems. 14, 51

UI User Interface. 24, 30, 32, 41, 45, 46, 61

URL Uniform Resource Locator. 56

UTM Unmanned Traffic Management. 3, 5

UX User Experience. 45, 46, 61

Appendix B

SD card contents

```
readme.txt ............... the file with SD card contents description
src ...................................... the directory of source codes
  thesis .............. the directory of LaTeX source codes of the thesis
text ......................................... the thesis text directory
  DP_Matějka_Jan_2020.pdf ............ the thesis text in PDF format
```