FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Implementation of the new module into the Dronetag web application for planning, managing and coordinating drone fleets
Student:	Michal Skipala
Supervisor:	Ing. Lukáš Brchl
Study program:	Informatics
Branch / specialization:	Web and Software Engineering, specialization Software Engineering
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2022/2023

Instructions

There is currently a lack of applications on the market that would allow drone pilots to conveniently manage and coordinate several aircraft within a single organization (drone fleet management). Most of the solutions are focused on only one drone manufacturer, have insufficient flight planning capabilities, or are simply overpriced. This thesis aims to utilize existing building blocks of the Dronetag web platform that already offers some drone coordination-related functionalities and extend for organization and fleet management use.

- Research, analyze and compare existing drone fleet management solutions.
- Define the functionalities of the planned fleet management module and design the user interface.
- Implement the functionalities with state-of-the-art web technologies and in accordance with the latest trends.
- Test the application with real pilots, evaluate the results and suggest its future improvements.

Electronically approved by Ing. Michal Valenta, Ph.D. on 8 February 2022 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Implementation of the new module into the Dronetag web application for planning, managing and coordinating drone fleets

Michal Skipala

Department of Software Engineering
Supervisor: Ing. Lukáš Brchl

May 12, 2022

Acknowledgements

I would like to thank all my friends and family for being supportive about my thesis. I would particularly like to thank my supervisor Ing. Lukáš Brchl for all the time he gave me during our consultations. Lastly, I would also like to thank the whole Dronetag team.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 12, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Skipala, Michal. *Implementation of the new module into the Dronetag web application for planning, managing and coordinating drone fleets.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstract

This bachelor thesis focuses on design and implementation of a fleet management module to existing application Dronetag using modern frontend web technologies – mainly TypeScript and React. The module allows users to form organizations, manage the organization, and plan missions.

The thesis is divided into smaller specific goals. First, we conduct research in which existing solutions on the market are analyzed and their advantages and disadvantages are listed. The following chapter Module Analysis puts focus on software requirements and use cases and defines what the module should do. Module Design chapter places a lot of emphasis on graphic and software design, describes the architecture, and provides a high-level overview of the designed solution. The implementation part of the thesis explains specific implementation decisions, introduces the libraries used in the project, and elaborates on more advanced components. The last chapter describes the testing process and provides a report on actual testing with real pilots and users. The last section of the chapter suggests future improvements to the module.

Keywords drone, fleet management, React, web application, front end, pilot, JavaScript, TypeScript, Dronetag

Abstrakt

Tato bakalářská práce se zaměřuje na implementaci modulu pro správu flotily do existující aplikace Dronetag použitím moderních frontend webových technologií – převážně TypeScript a React. Tento modul umožňuje uživatelům zakládat organizace a plánovat mise.

Práce je rozdělena na několik podčástí. Nejprve přichází rešerše, kde jsou analyzována existující řešení na trhu a shrnuta jejich pozitiva a negativa. Následující kapitola se zaměřuje na softwarové požadavky a případy užití, které by měl modul splňovat. Kapitola Module Design klade důraz na grafický a softwarový návrh, popisuje architekturu a dává popis řešení na vysoké úrovni abstrakce. Implementační část mé práce vysvětluje spoustu konkrétních rozhodnutí a představuje knihovny, které byly pro vývoj modulu použity. Poslední kapitola popisuje proces testovaní a poskytuje report z testování s opravdovými piloty a uživateli aplikace. Poslední sekce této kapitoly pak navrhuje vylepšení a změny, které by bylo vhodné realizovat do budoucna.

Klíčová slova dron, správa flotily, React, webová aplikace, front end, pilot, JavaScript, TypeScript, Dronetag

Contents

List of Figures

List of Tables

Introduction

Number of companies that use drones on a daily basis is growing rapidly. Drone missions are becoming more complicated, and it is not an easy task to carry them out in an organized manner.

There are many types of missions that companies need to carry out. Ranging from geographical mapping, such as capturing an area of ground for survey or construction work supervision, to dangerous tasks such as fire fighting, accident investigation, or cave exploration, it is becoming harder to control, and missions can no longer be flown by eye alone. These missions are usually carried out by a team of professional drone pilots.

There is a market demand for a web application that would serve drone pilots as a tool to manage their large drone fleets, plan complex missions, and review them retrospectively.

Motivation

I chose to work on this thesis because it is a perfect opportunity to apply my knowledge of software engineering on a real project. Dronetag is a successful project, and its products are used by numerous drone pilots all around the world. The number of companies that use drones to fulfill their business case is increasing every day, but there are few software solutions on the market for managing drone fleets and planning drone missions.

Another reason was to gain valuable experience in state-of-the-art frontend web technologies. I am very interested in this area of software development, and there are not many opportunities to go in-depth on this problem domain while studying. I believe that this thesis could not only meet the increasing demand for a comprehensive drone fleet management tool but also work as a study material for aspiring frontend React web developers. I will be happy to explain all the important aspects of this JavaScript library and present what I consider to be the cleanest and most modern React software architecture.

Objectives

Due to the wide scope of this thesis, there are smaller specific goals that collectively fulfill the main objective.

Main Objective

The thesis aims to develop a module into the existing Dronetag application. The module will allow users to form organizations. In these organizations, users will be able to share their aircraft, Dronetag devices, and plan various drone missions. This module will fit into the existing Dronetag code base and follow the best practices of React frontend web development.

Specific Objectives

1. Research, analyze, and compare existing drone fleet management solutions on the market and describe their strengths and weaknesses.
2. Define the functionalities of the planned fleet management module and design the user interface, applying software engineering concepts and following best practices.
3. Implement the defined functionalities with state-of-the-art web technologies in accordance with the latest trends.
4. Test the application with real pilots, evaluate the results and feedback, suggest its future improvements.

Chapter 1

Existing Solutions

This chapter will focus on the analysis of existing drone fleet management solutions on the market. Each solution will be analyzed, evaluated, and its strong and weak points will be listed.

1.1 AirHub

AirHub is the first analyzed platform. It is an all-in-one solution offering two applications: Drone Operation Center for the web and Ground Control App for the mobile. The following sections will focus on the Drone Operation Center and its functionalities.

Pricing

AirHub [1] is a commercial platform that offers different functionality depending on your subscription plan. The functionalities of the AirHub enterprise tier were demonstrated to me by an AirHub employee on a video call.

Free tier allows the user to view the airspace zone, weather forecasts, advisories, permission forms, use the mission planner in the mobile app, and write notes. Users can manually log flights from the mobile app, fly with Da-Jiang Innovations (DJI) drones, and manage personal documents.

Pilot tier unlocks additional features such as creating checklists, Unmanned Traffic Management (UTM) connection, logging incident, seeing media on the map, flight analytics, file and asset management. It costs 10 euro per month.

Enterprise tier provides access to all functionality of the platform: automated flights, live video stream, team management, team documents, and live airspace management. The price is not fixed and potential customers must contact AirHub for negotiations.

Features

Drone Operation Center offers many features compared to other drone web platforms on the market even in the free tier.

Dashboard is the main page of the web application. It shows live airspace, incidents, user profile, user activity in the form of notifications, maintenance, recent media, and media on the map. See image 1.1.

Library allows the user to add his drones. The user can specify basic information: name, model, manufacturer, serial number, weight, registration, firmware version, and picture. After clicking on the drone, the user can see detailed information including general information, logbook (past flights), maintenance, documents, and statistics. In addition to drones, the user can manage his batteries, equipment, checklists, documents, and media in the library.

Logbook lists the user's flights that were flown with the AirHub application. The user can export the flights to PDF and CSV or import the flights from a DJI drone. It has a search bar and allows users to filter flights by date. After clicking on the flight, a flight detail pops up, revealing all the important information about the flight. See image 1.3

Live airspace page shows a full-screen map with active mission sidebar.

Incidents is the page where the user can view incidents and filter them by date, status, and type. Incidents have a name, description, location, investigator, reporter, related flight and status.

Maintenance page allows the user to add a maintenance action and list past maintenance. The user needs to select an asset and fill in the maintenance details: inspection name, deadline, costs, technician, and notes. On the maintenance edit page, the maintenance status can be set from planned to in progress, completed, partially completed, or postponed.

Teams is a feature that is only available to enterprise users. The page allows eligible users to form and manage teams. Team members have the following roles: administrator, pilot, observer, and payload operator. The detail shows the flight statistics aggregated over the whole team: amount of flights, average flight time, and total flight time.

Flight planning allows the user to plan flights by selecting a flight area and naming the flight. It is possible to add team members into the flight and select a drone from the library to fly with.

Evaluation

AirHub is a modern looking application, and the Drone Operation Center is able to help many companies carry out their drone missions. With comprehensive flight statistics and features such as team management, maintenance logging, and a powerful library that can track many different types of assets, it is a very universal solution for drone fleet management.

Strong Points

- Clean and modern looking user interface,
- Working solution for organization fleet management,
- Handling asset maintenance in the application could be very useful to companies,
- Application provides team statistics.

Weak Points

- Library page handles too much asset management,
- Assigning roles directly to team members is not very flexible,
- No option for ”teams within a team”,
- Team management only in enterprise tier.

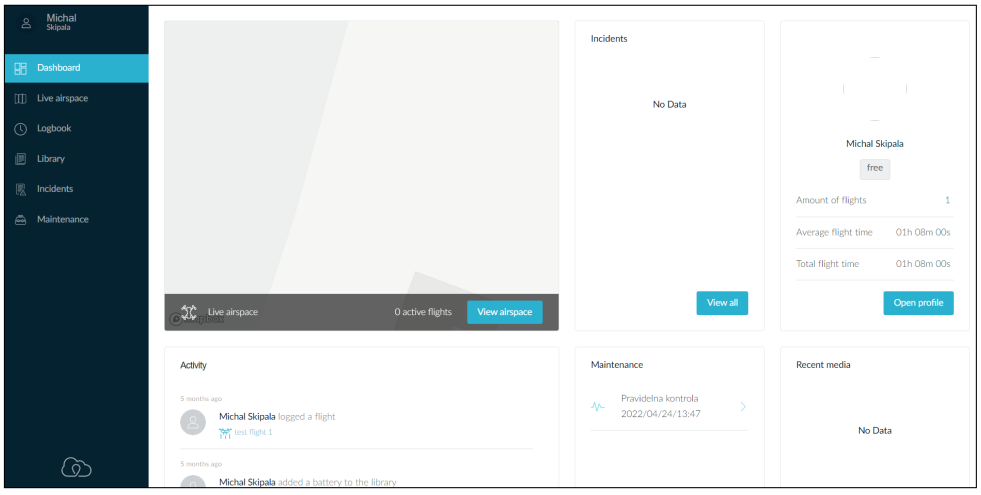

Figure 1.1: AirHub Dashboard [2]

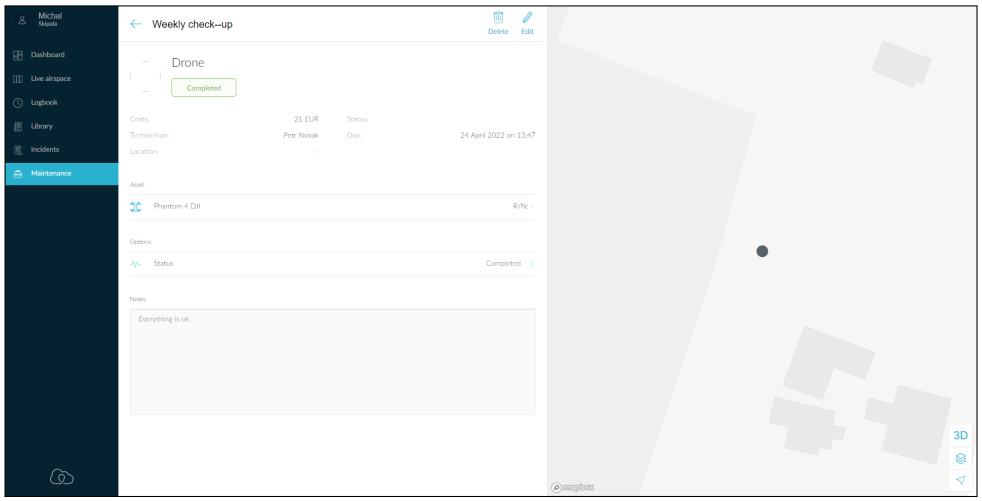

Figure 1.2: AirHub Maintenance Detail [2]

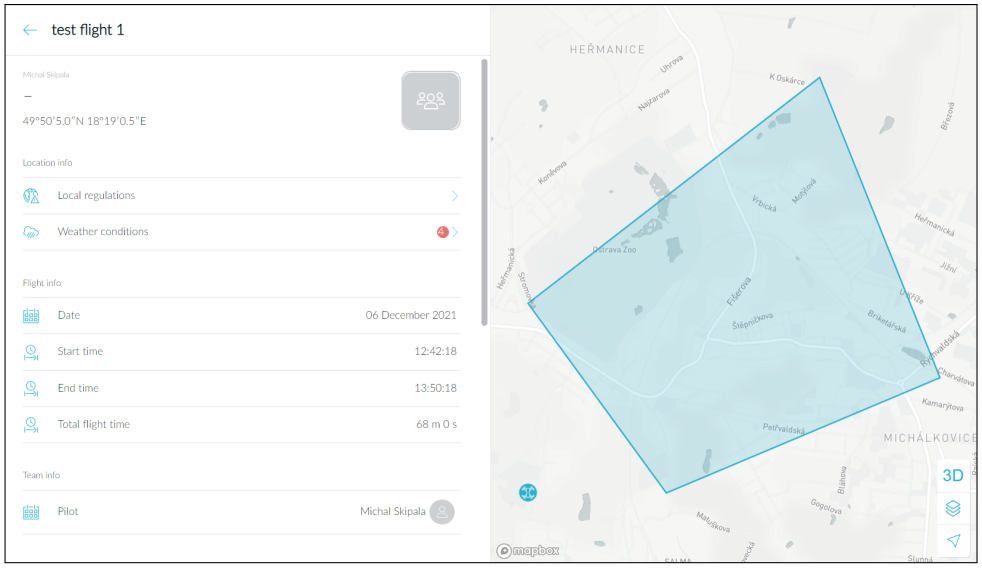

Figure 1.3: AirHub Flight Detail [2]

1.2 DJI FlightHub Enterprise

DJI FlightHub [3] was released in 2017, but it seems that the majority of the focus was on the development of FlightHub 2 with a planned release in 2022/2023.

Installation

DJI FlightHub Enterprise is an application that needs to be installed on a server. To install the application, the user needs to set up Ubuntu Server 16.04 and install it there. A detailed tutorial can be found at the DJI FlightHub Download Center [4].

Pricing

DJI FlightHub Enterprise is a commercial platform that offers three levels of subscription. More detailed pricing can be found on the DJI website [3].

Basic allows the user to bind up to 5 drones with all features and is priced at $99 USD per month or $999 USD annually.

Advanced allows the user to bind up to 10 drones with all features, with the price reaching $299 USD per month or $2999 USD annually.

Enterprise allows binding of more than 10 drones and provides the ability to integrate data into a private cloud. The price is not specified and is subject to negotiation.

Features

Live operations section shows the drone location in real-time, and a live video of up to 4 drones on one page. See image 1.6.

Data statistics allows the user to look at team or pilot statistics: flight time, travel distance, top distance, top speed, and more. The user can also see the flight routes. See image 1.4.

Mission planning is the page where users create missions. It is possible to plan a waypoint mission – set waypoints, aircraft speed and altitude, and points of interest, specifying more information for each waypoint, such as yaw or gimbal control – or a mapping mission. See image 1.6.

Team management section enables the user to create members and teams. Members can be administrators, captains, and pilots. Administrators invite other captains and pilots to the team. Captains can only invite pilots. Both administrators and captains can access the FlightHub Enterprise platform, while pilots can only execute missions, enable the live

function, upload data, bind devices, and view information in the mobile application.

Evaluation

DJI FlightHub Enterprise could be an interesting drone fleet solution for companies that need to automate their drone missions. The application allows users to fly only with DJI drones, which could be a dealbreaker for many companies, the installation is difficult, and the user interface is not intuitive.

Strong Points

- Working fleet management solution,
- Waypoint missions – interesting concept and an efficient way to automatize simple and repetitive missions.

Weak Points

- User interface does not look modern,
- Very difficult to configure for users without IT knowledge,
- Could be too expensive for a smaller company,
- Very limited supported aircraft (DJI Matrice series).

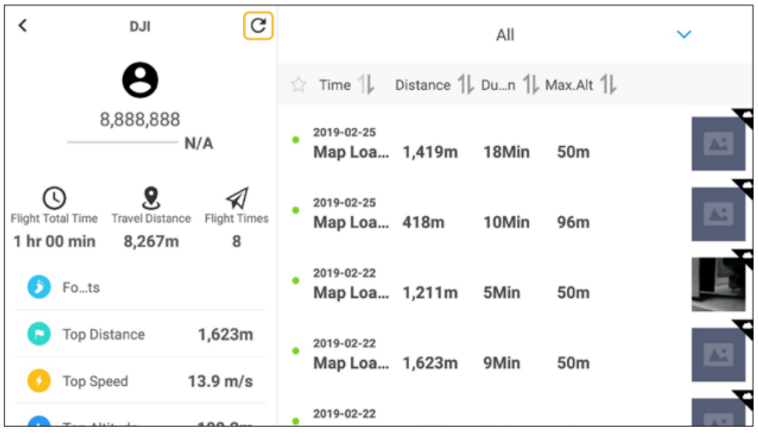

Figure 1.4: DJI FlightHub Statistics [5]

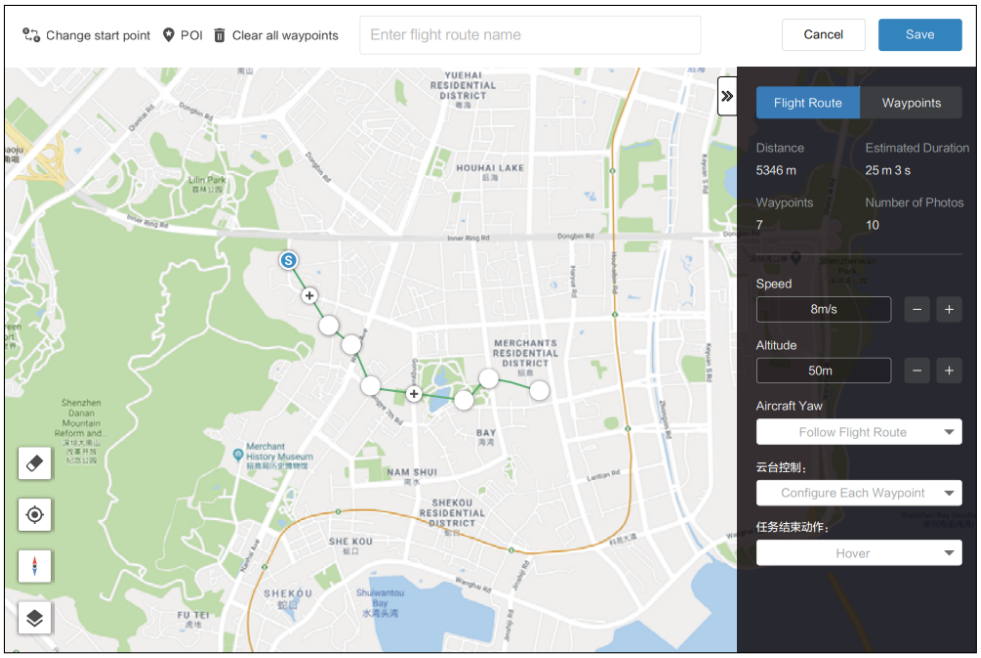

Figure 1.5: DJI FlightHub Mission Planner [5]

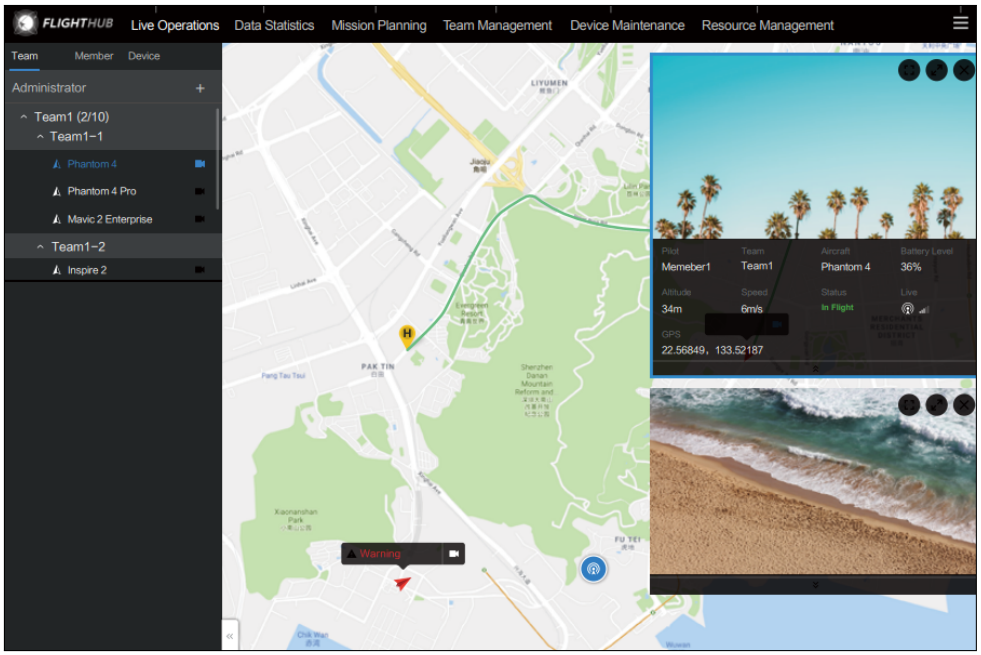

Figure 1.6: DJI FlightHub Live [5]

1.3 Other Solutions

There are other drone fleet management solutions on the market, but they were left out of this thesis due to the limited available information on the features of the solutions and unsuccessful requests for a demo, which would make the analysis very limited. To see more drone fleet management platforms analyzed, refer to thesis *"Implementation of the new module into the Dronetag Mobile app in Flutter for planning, managing and coordinating drone fleets"* by Albert Moravec to appear in 2022, where two other interesting drone fleet management platforms will be analyzed.

Chapter 2

Module Analysis

In this chapter, the Dronetag platform will be introduced, the planned module will be analyzed, the software requirements will be defined, and specific use cases will be listed. To correctly specify the functionalities of the planned module, it is necessary to first analyze the existing Dronetag platform. The next section will introduce Dronetag and describe each of the application building blocks and technologies that were used.

2.1 Dronetag

Introduction

Dronetag [6] is a team of enthusiastic developers, aviation experts, and electrical engineers. The project came to life in 2018 at the Space Technology Hackathon. In the following years, Dronetag has kept growing and is now offering an all-in-one solution for drone pilots.

Products

The main product offered by Dronetag is *Dronetag Mini* device. It is a small device that can be mounted onto any drone and identifies the drone in the air. It works by getting the drone coordinates from Global Positioning System (GPS), Global Navigation Satellite System (GLONASS), Galileo, and European Geostationary Navigation Overlay Service (EGNOS) satellites and sending the coordinates and drone ID to the central system using mobile network. The device communicates with the surrounding devices via Bluetooth.

Dronetag also offers a web application and multi-platform mobile app. These platforms were designed primarily for a single user or a small team, and they are not that convenient for large organizations.

Drone Fleet Management

The objective of this thesis is to develop a module for the existing Dronetag application that would extend it. The module will embody a drone fleet management and mission planning solution for large organizations. This thesis focuses mainly on the desktop web platform. The core functionalities of the existing web application will be described in the following section.

2.2 Dronetag Web Application Overview

The main functionalities of the web application are flight planning, asset management, and flight data manipulation. The following sections will demonstrate the existing features of the application.

Aircraft management

The application allows the user to manage his drones. The user can add a drone and specify the name, model, weight, aircraft class, and endurance. Users can view their drones in a filterable list, edit their properties, or remove them entirely. Aircraft are used mainly for flight planning and filtering flights by aircraft.

Device management

Users can add their own Dronetag devices. The devices are linked to user accounts by their serial number. In the list view, the devices are shown as clickable cards. Clicking on a device opens the device detail page.

The detail shows the serial number and current status: connectivity information and battery life. It is possible to edit the device name or remove it from the account. See image 2.2.

Flights

The user can view all past, planned, and canceled flights in the Flights section. The flights can be filtered by type, device, or aircraft. The flight detail shows the flight information, and past flights reveal the flight path of the drone. See image 2.3.

Settings

The user is able to change his credentials, default aircraft, identification, profile details, User interface (UI) preferences, notifications and Application programming interface (API) keys in the Settings section.

Flight Planner

The main feature of the application is the planning of drone operations in Flight Planner. Flight planning is accomplished by setting a take-off point and drawing a flight region, which can be a polygon or a circle drawn on the map. The flight region describes the boundaries of the flight.

After drawing the flight region, there are multiple things that need to be assigned:

- Height range – lower and upper altitude of the flight,
- Aircraft – aircraft used for the flight,
- Device – Dronetag device used for the flight,
- Start and finish dates – planned start date and finish date,
- Flight visibility – public or private,
- Operation mode – Visual line of sight (VLOS) or Beyond visual line of sight (BVLOS),
- Operation category – open, specific, or certified.

Once the flight plan is completed, the Dronetag application checks for possible collisions with other flight plans. If no collisions are found, the flight is created. See image 2.1.

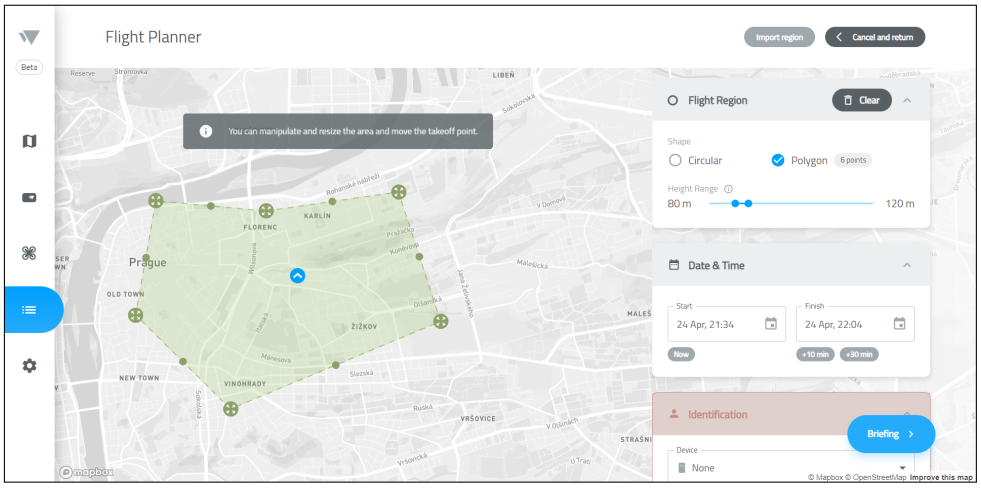

Figure 2.1: Dronetag Flight Planner [7]

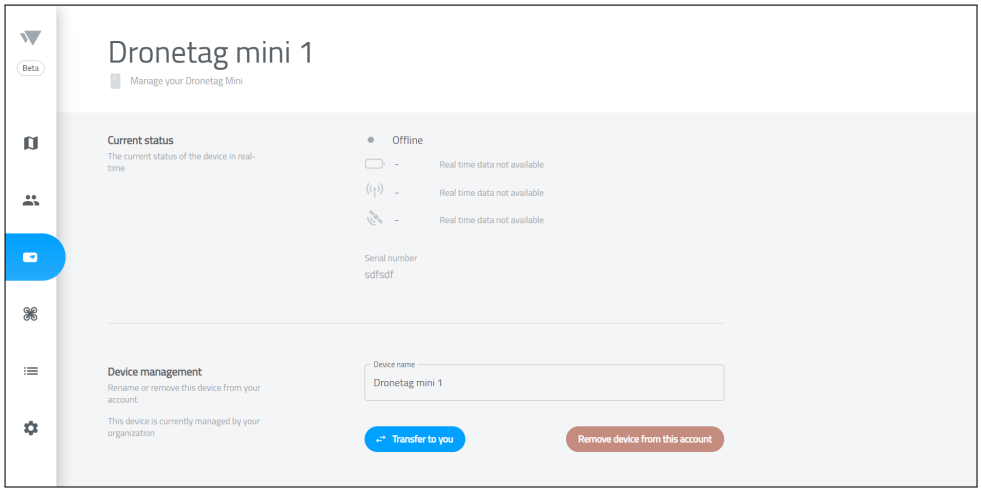

Figure 2.2: Dronetag Device Detail [7]

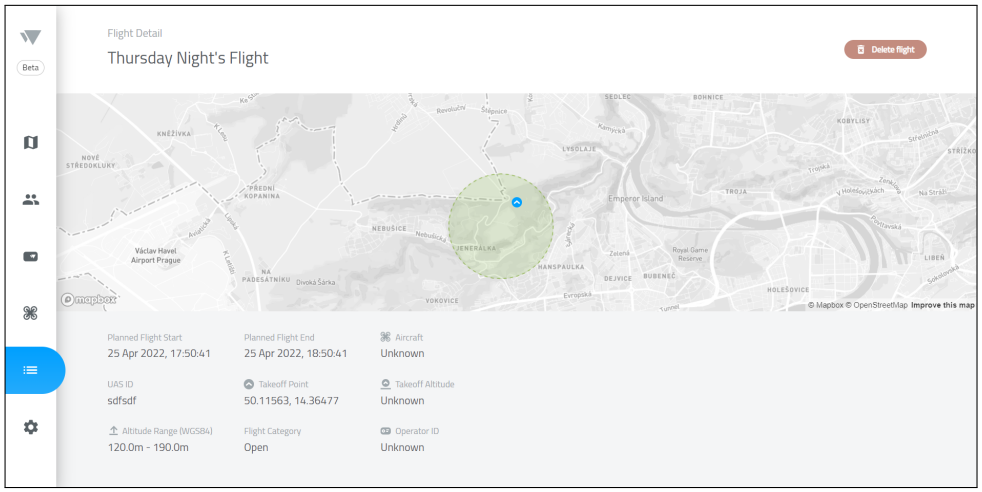

Figure 2.3: Dronetag Flight Detail [7]

2.3 Application Architecture

Dronetag application consists of a backend service, PostgreSQL [8] database, live service, frontend web application and mobile app. The services together form the Dronetag platform and communicate with each other. To see a visual representation, please refer to image 2.4. Since the thesis focuses mainly on the web application, the services and the mobile app will only be referenced and briefly described in the following sections.

Backend Service

Backend service was written in Python [9] using Django [10], Django REST framework [11] and Celery [12]. The main responsibility of the service is to handle requests coming from the frontend web application, mobile app, and live service. It also provides access to the Dronetag database and handles the database manipulation.

Live Service

Live service serves as a real-time data provider, broker, and distributor. The service is used for data interchange between Dronetag devices and client components via Hypertext Transfer Protocol (HTTP) Representational state transfer (REST) and Websocket technologies. It enables clients to display frequently changed data (telemetry data, device update, current status) in real time, effectively, and with low latency.

Mobile App

Mobile app is a multi-platform mobile application for iOS and Android. It is written in Dart [13] using the Flutter [14] framework and UI toolkit. The architecture uses the BloC [15] library for the state management of the app. It communicates with the back end via the HTTP REST protocol.

Web Application

The web application was written mainly in TypeScript using the React library. It communicates with backend service and live service via the HTTP REST protocol. The technologies that were used to build the web application will be described in more detail in the next section.

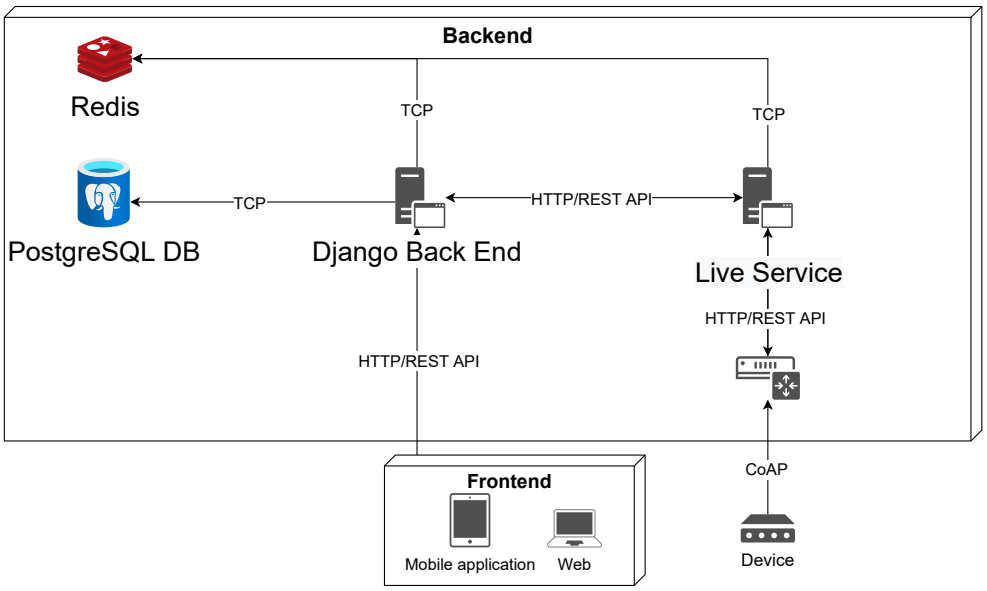

Figure 2.4: Dronetag Current Architecture

2.4 Web Application Technology Stack

In this section, the technologies used on the front end will be introduced and thoroughly analyzed. The planned module will be built on these technologies to integrate correctly into the existing codebase.

JavaScript

JavaScript [16] is a compiled, interpreted Just in Time (JIT) programming language. It supports object-oriented, imperative, and declarative styles. The language standards are defined in ECMAScript 2023 [17], which also serves as a comprehensive and detailed JavaScript documentation.

JavaScript is mainly used to add behavior and special effects to HyperText Markup Language (HTML) and Cascading Style Sheets (CSS) web pages. It enables the user to interact with the web page.

There are many libraries and frameworks built on top of JavaScript, which help programmers develop complex and multi-layered web applications while still allowing the developer to maintain a clean and extensible architecture.

TypeScript

TypeScript [18] is a strongly typed programming language that builds on JavaScript, adds additional syntax, and provides JavaScript with type infer-

ence. It is preferred to use TypeScript instead of plain JavaScript because the type inference prevents many bugs that could possibly be caused by developers. It is widely supported by JavaScript libraries and runs anywhere JavaScript runs. Some of the main advantages of using TypeScript instead of plain JavaScript include:

- Static typing – makes it impossible for a declared typed variables to change its type, which prevents a lot of errors that could be hard to be find and reveals many bugs before compilation,
- Type inference – which provides helpful code suggestions based on the type of variable or object and eliminates the need to jump back and forth between files to check the type,
- Object-oriented concepts – TypeScript extends JavaScript by adding important Object Oriented Programming (OOP) concepts, such as interfaces.

React

React [19] is a free and open-source JavaScript library for building web applications and user interfaces. The core concept of React is separation of concerns by building the program from loosely coupled units called components. React works perfectly with TypeScript, which enables static type checking to prevent possible problems related to types.

Since React is not opinionated about the build and structure of the application, every developer writes React programs a bit differently, which might seem chaotic. It is important to maintain a consistent architecture throughout the codebase when working in team.

React is the main library used to build the Dronetag front end and it is necessary to explain its main concepts. The following sections will further elaborate on the React library.

JSX

JavaScript XML (JSX) [20] is a syntax extension to JavaScript that allows writing HTML within JavaScript code. JSX is an expression and becomes JavaScript at runtime. That enables the use of JSX inside if statements and for loops, assigning it to variables, returning it from functions, and accepting it as a function argument, providing great flexibility.

The structure of JSX looks very similar to HTML but allows embedding JavaScript expressions into HTML, such as ternary operators. That is very useful for conditional rendering. While it is not necessary to use JSX in React, it is widely used and recommended. JSX produces React elements that are later rendered into Document Object Model (DOM).

DOM

DOM [21] defines a standard for accessing documents. HTML DOM is a programming interface and object model for HTML. It is constructed as a tree of objects. It is used to get, change, add, or delete HTML elements to a web page.

Virtual DOM

Virtual DOM [22] is a virtual representation of the actual DOM kept in memory. Every time the state of the React application changes and the DOM needs to be updated, the Virtual DOM is updated instead. The Virtual DOM is then compared to the browser DOM, the difference between them is computed, and the Virtual DOM calculates the best possible method to make the desired change to the DOM. This process significantly increases web performance because it ensures minimal manipulation with the actual DOM.

React listens to changes in its components using the observer pattern [23] and finds out which Virtual DOM objects have changed. The performance is significantly better compared to manipulating the DOM directly. Image 2.5 helps visualize the concept of the Virtual DOM.

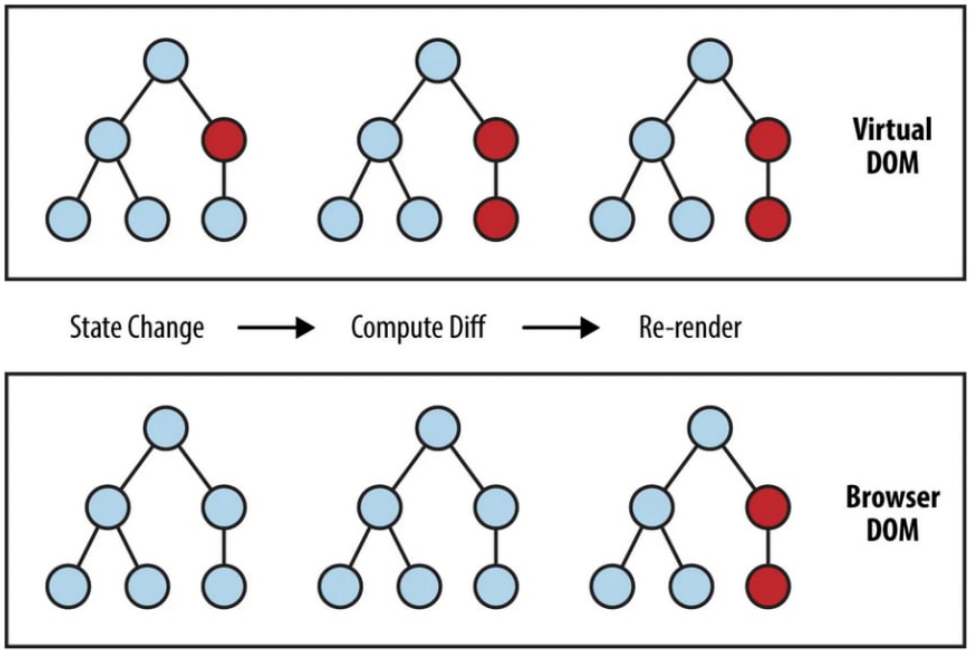

Figure 2.5: Virtual DOM [24]

React Components

React applications are built by splitting the program into independent, loosely coupled and reusable pieces of code called components [25]. Components accept props as an argument. There are two types of components:

Function components – plain JavaScript functions that accept a single argument called props with data that the components need and return a React element,

Class components – JavaScript ES6 classes that extend React.Component. These components have a state and provide component lifecycle methods described in the following section.

There are stateful and stateless React components. Stateless components are mostly used only to render the data passed down to them. Stateful components usually handle business logic and store data in a local state.

Every React component must act like a pure function. That means it needs not to change its props, and the output value should be the same for the identical input. There are ways to implement stateful logic to function components without violating this rule using React Hooks that will be described later in this chapter.

React Component Lifecycle

Each component in React has a lifecycle [26]. The lifecycle is divided into three phases:

- Mounting – when the component is put into the DOM,
- Updating – when the component is updated and a change occurs in its state or props,
- Unmounting – happens when the component is removed from the DOM.

See image 2.6 for a graphic demonstration.

React Data Flow

React has a unidirectional data flow [28]. The state can only be passed down the component tree, and any changes in a parent component should only affect its children. If the state of a parent component needs to be changed from its child, it is possible to ”lift the state up” [29] by passing down a function from the parent component. The state change happens inside of the parent, and the child only ”initiates” the change.

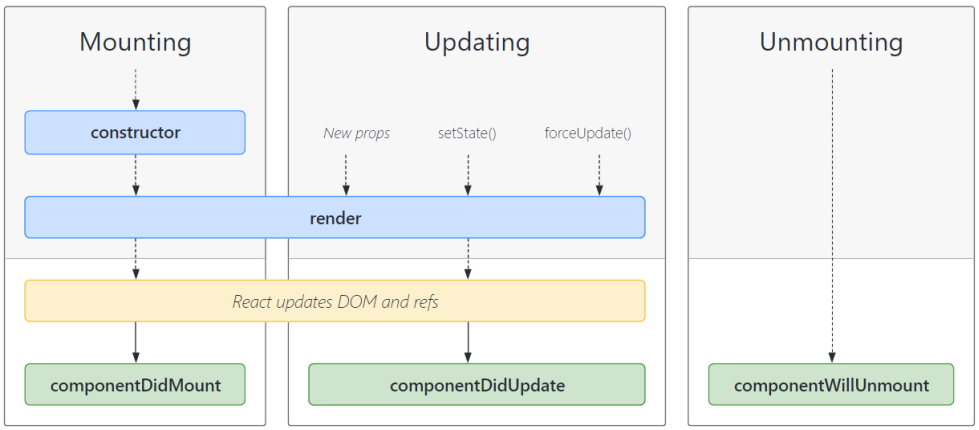

Figure 2.6: React Lifecycle [27]

React Hooks

React Hooks [30] were introduced in React 16.8 in 2019. They allow the use of state in function components and provide the ability to reuse complex logic between components. Hooks are very useful for local state management, data fetching, and isolating complex business logic. There are important rules that Hooks should always follow:

- Call Hooks at the top level before any returns,
- Call Hooks from React functions, not from JavaScript functions,
- Do not call hooks conditionally or inside loops.

React heavily depends on the order of the Hook calls, and these rules make sure that the Hooks are called in the same order in every render.

2.5 Functional Requirements

This section will describe the functional requirements of the fleet management module. These requirements will define the expected fleet management functionalities and behavior. The requirements presented in this section are a combination of current Dronetag user requests, research of existing solutions on the market, and discussion between the author and other members of the Dronetag team developing the application.

Functional requirements [31] describe the intended behavior of the system. You could imagine this behavior as tasks, services, or functions that the system needs to perform.

As Wiegers [32] had written:

> *"Software developers don't implement business requirements or user requirements. They implement functional requirements, specific bits of system behavior."*

Due to this reason, it is extremely important to correctly capture the functional requirements. Any further development directly depends on them.

FR01 Organization Management

Any user that is not in an organization will be able to create a new organization. The user who creates an organization will become its owner. The owner will be able to change the organization name and description, change the global organization settings, and delete the organization. If the organization is deleted, all of the organization data will be lost.

FR02 Organization Asset Management

Any member of the organization will be able to transfer his aircraft or device to the ownership of the organization. Any member will also be able to transfer any organization aircraft or device into his ownership.

FR03 Organization Member Management

Any organization member will be able to invite new members to the organization using their email address. The invited user will be able to accept or decline the invitation via a direct link or in the Dronetag web app in case they already have an account and are not a member of an organization. Any organization member will be able to leave the organization. Any organization member can be removed from the organization by the organization owner. If organization member leaves the organization, all of his data related to the organization will be lost.

FR04 Organization Team Management

Team is a list of selected organization members. Any organization member will be able to create a team, rename the team, add other members of the organization to the team, or delete the team.

FR05 Mission Management

Any organization member will be able to create a mission. The member who created the mission will become the mission creator and mission coordinator. The mission coordinator can add members to the mission, remove members

from the mission, and assign pilot and coordinator roles to members in the mission. The mission coordinator will be able to see all flight plans of the mission and assign or edit flight plans to all mission pilots. The mission coordinator cannot be removed from the mission. The mission member with pilot role will be able to create and delete his own mission flight plan. The mission pilot will not be able to edit the mission plan or any other flight plans. Mission member that is neither pilot nor coordinator will be able to view the mission, but will not be able to make any changes to the mission.

2.6 Nonfunctional Requirements

Nonfunctional requirements [33] describe the global requirements on development, operational costs, performance, reliability, maintainability, portability, and robustness that the final solution is expected to meet. They play a critical role during system development and have considerable influence on design and implementation decisions. All of them have to be accounted for, which is why they will be given a code and description.

NFR01 Functionality will be available via HTTP REST API

To ensure that the application communicates correctly with the back end, it will use the HTTP protocol and respect the REST constraints.

NRF02 Functionality will be divided into multiple components

Separation of concerns is an important concept that every scalable enterprise application should follow. The application will be split into multiple components, each of them focused on one thing.

2.7 Use Cases

Use cases [31] define a set of interactions and actions between external actors and the system. Actors are all parties outside the system that interact with the system.

A use case is initiated by the actor with a particular goal that he wants to reach. The use case is completed when the goal is reached. We can think of the use case as the sequence of interactions between the actor and the system that is needed to complete the use case.

Use cases allow for a more detailed and accurate description of the desired outcome. They are used to capture functional requirements in detail and provide an easier way to verify that all the requirements have been taken into account. Use cases are usually written in an easy-to-understand and structured narrative, making it easier for users to verify them. See table 2.1 for a use case model of the module.

Due to the large number of use cases and their nature being mostly simple CRUD operations, they will merely be listed.

Organization Management

UC01 Create organization

UC02 Rename organization

UC03 Delete organization

UC04 Leave organization

Organization Asset Management

UC05 View organization assets

UC06 Transfer asset to organization

UC07 Transfer asset from organization

Organization Member Management

UC08 View organization members

UC09 Invite member to organization

UC10 View pending invitation

UC11 Accept invitation to organization

UC12 Decline invitation to organization

Organization Team Management

UC13 Create team

UC14 Rename team

UC15 View team members

UC16 Add team member

UC17 Remove team member

Mission Management

UC18 Create mission

UC19 Edit mission

UC20 Assign flight plan to mission pilot

UC21 View missions

UC22 Delete mission

	Requirements				
Use Case	FR01	FR02	FR03	FR04	FR05
UC01	+				
UC02	+				
UC03	+				
UC04	+				
UC05		+			
UC06		+			
UC07		+			
UC08			+		
UC09			+		
UC10			+		
UC11			+		
UC12			+		
UC13				+	
UC14				+	
UC15				+	
UC16				+	
UC17				+	
UC18					+
UC19					+
UC20					+
UC21					+
UC22					+

Table 2.1: Use Case Model

Chapter 3

Module Design

This chapter will focus on the design of the planned module. The first section will introduce a graphic prototype that captures the pages of the module. The following sections will emphasize the software design of the module.

3.1 Graphic Prototype

When designing a web application, it is very helpful to create wireframes. Wireframes visualize page layout, functionalities, and intended behavior. Graphic prototype on the other hand, features the styling, fonts, colors, and other visual aspects of the page, as well as the functionalities and behavior.

Due to the complexity of this module, it was concluded that a graphic prototype would be more appropriate compared to wireframes and should be the first step to take during the software design of the module. The following subsections will showcase the graphic design of the selected pages. The interactive graphic prototype was made with a powerful prototyping tool Adobe XD [34]. The graphic prototype was made right after gathering the functional requirements. It helped visualize all the possible use cases, and revealed many edge cases that the software design had to later account for.

Aircraft Page

User should be able to see aircraft that belong to the organization and transfer aircraft between his account and the organization. It was decided to implement the transfer functionality in the Aircraft Edit Page. See image A.1.

Teams Page

User should be able to view organization teams, add a new team, add organization members to the team, remove organization members from the team, and edit the team. See image A.2 and image A.3.

Members Page

The first page the user should see is a list of members. It is the default page after you switch to organization from the side bar. The user should be able to invite new members to the team, and the organization owner should be able to remove other members from the organization. See image 3.1.

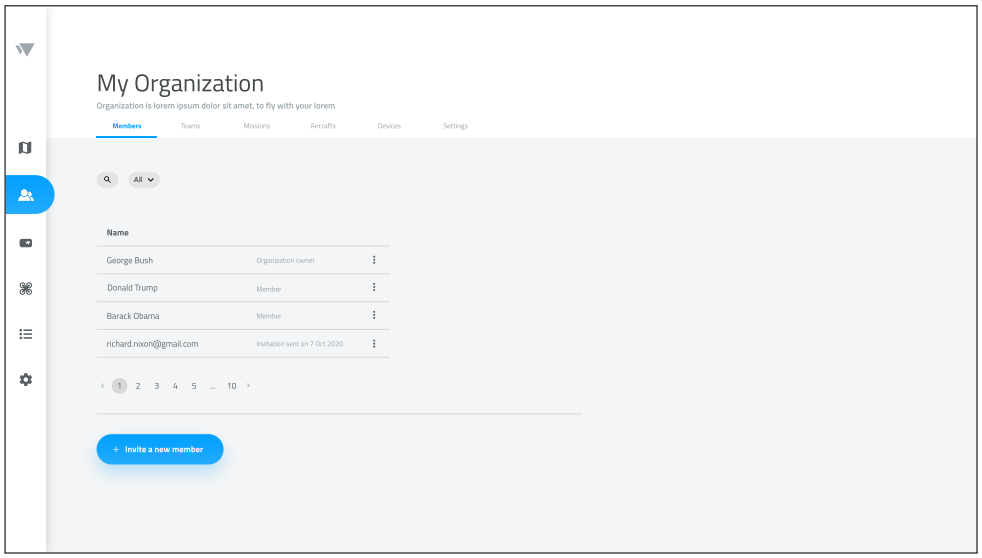

Figure 3.1: Organization Members Page

Devices Page

User should be able to see the devices that belong to the organization and transfer the devices between his account and the organization. It was decided to implement the transfer functionality in the Device Edit Page, which is already implemented. See image A.4.

Settings Page

Organization owner should be able to delete and edit organization details such as name, description, and preferences. The user should be able to leave the organization and check the organization settings. See image A.5.

Missions Page

User should be able to see a list of planned, current, and past missions. Mission should present its basic information – start date, end date, number of pilots, and name of the coordinator. See image A.6.

Mission Detail Page

User should be able to view a mission detail of a past mission. Mission detail should show all relevant mission information to the user. See image 3.2.

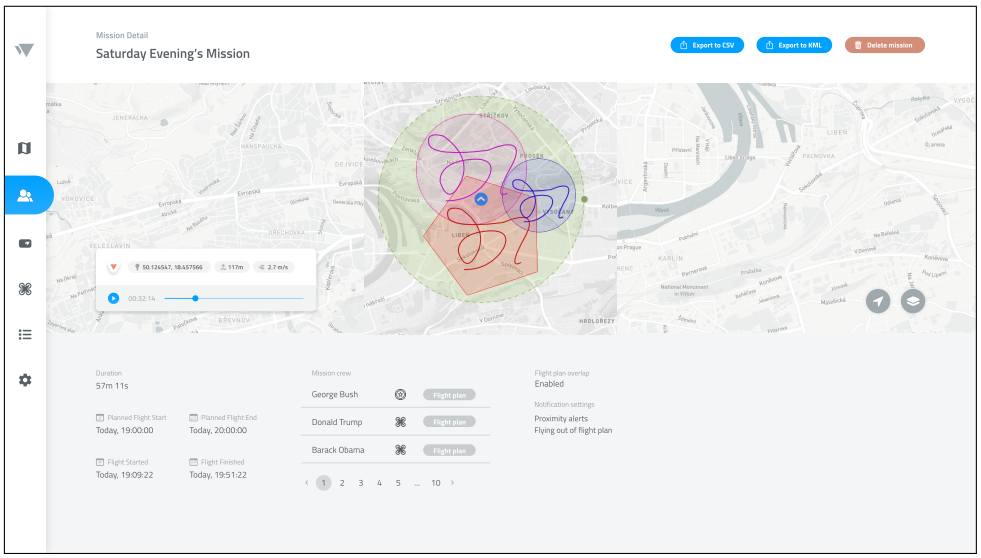

Figure 3.2: Organization Mission Detail Page

Create Organization Page

User will be able to create an organization if he is not a member of any organization. On this page, the user will also see all the invitations to existing organizations and will be able to either decline or accept these invitations. See image A.7.

Mission Planner Page

User should be able to plan a mission. The process of creating a mission is a complicated User experience (UX) problem and will be explained in the following sections. Due to this reason, mission planning had to be split into two steps. Designing the mission planner with these capabilities was very difficult. To see my original draft, refer to images 3.3 and 3.4.

Dronetag team decided to cooperate with UI/UX professionals from company StrideXL [35] to design a user-friendly mission planner with the specified functionality. After a few iterations and collective meetings, StrideXL designed a simple and intuitive mission planner. See images 3.5 and 3.6. In the following sections, the new mission planner will be used.

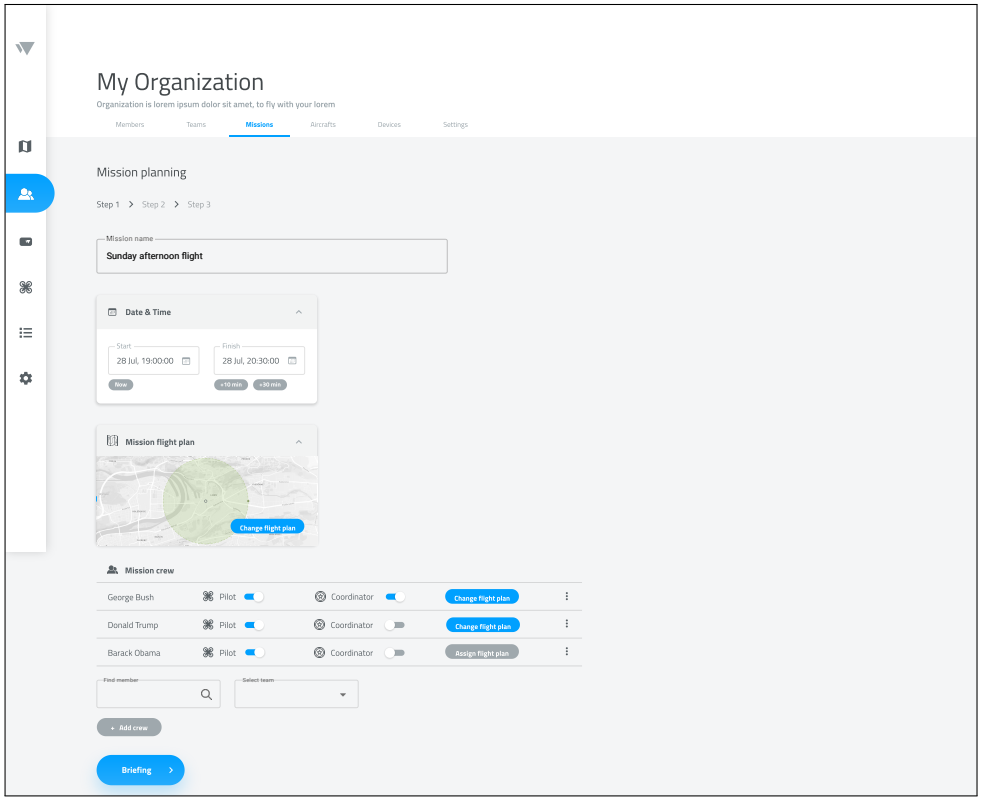

Figure 3.3: Old Mission Planner Page, Step One

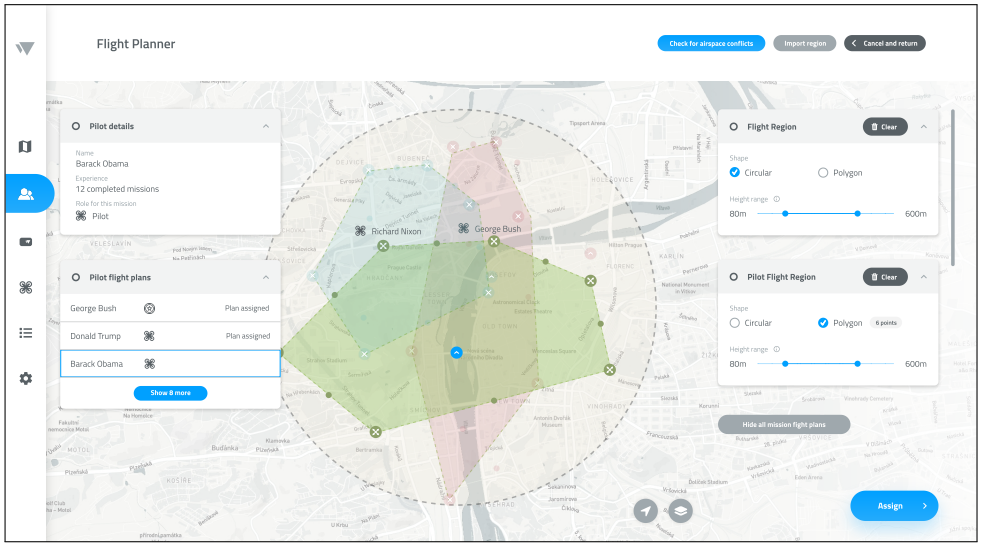

Figure 3.4: Old Mission Planner Page, Step Two

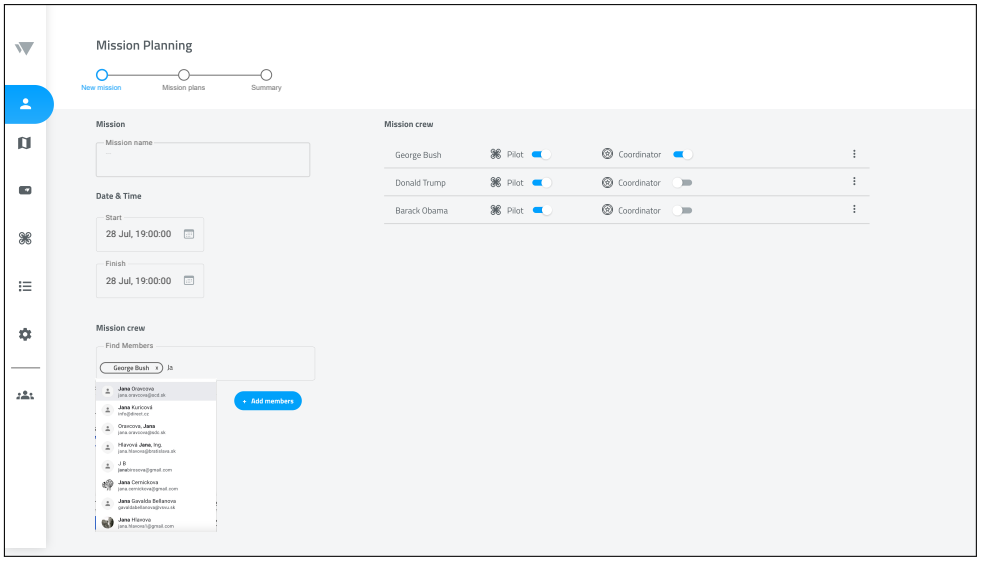

Figure 3.5: New Mission Planner Page, Step One

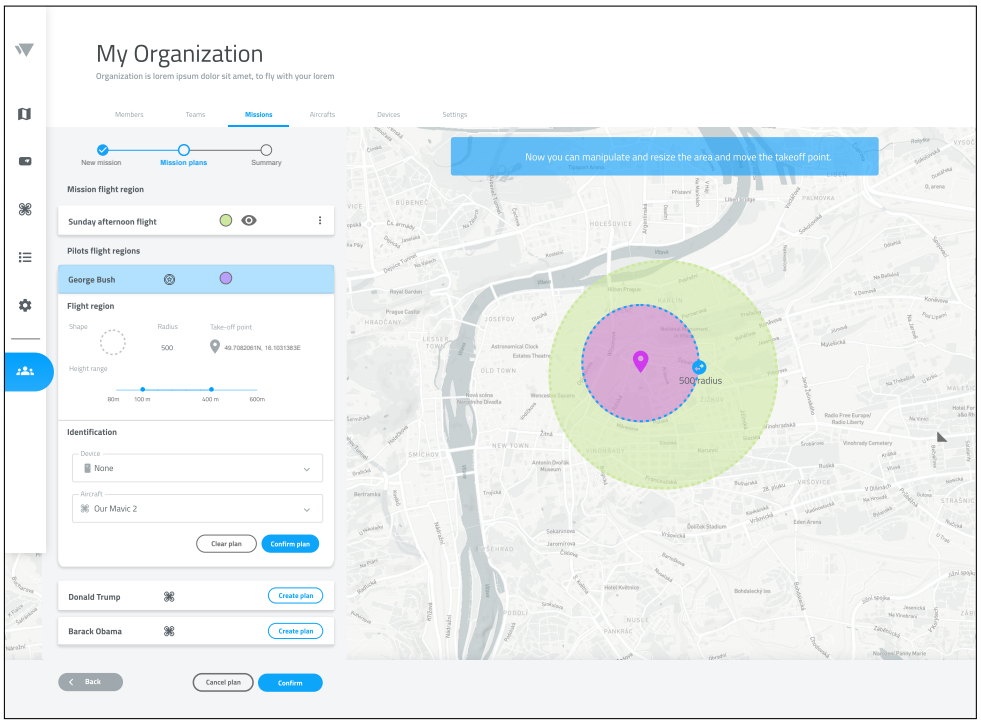

Figure 3.6: New Mission Planner Page, Step Two

New Mission

In the first step, the user will fill out a form containing general information about the mission: mission name, planned start date, planned end date. The user will also add all the organization members that should be involved in the mission and assign them their roles for the mission. See image 3.5.

Mission Plans

In the second step, the user will create a mission plan by drawing a shape on the map and providing the height range of the mission, defining the ”mission region”. The process is visualized in the activity diagram 3.8. After confirming the mission plan, the mission creator will be able to assign a flight plan to every pilot in the mission by drawing a shape on the map, specifying the height range, and selecting an aircraft and device for the pilot. This process is also visualized in diagram 3.8. The mission creator does not have to assign any flight plans to pilots.

Mission Edit Page

User will be able to view a planned mission. If he is a coordinator, he will have the ability to clear or create the mission plan. Clearing a confirmed mission plan will delete the mission. The coordinator will be able to clear or create any flight plan. Clearing a confirmed flight plan deletes the flight. If the user is a mission pilot, he will be able to clear or create his own flight plan. If he is neither pilot nor coordinator, he will be able to see the mission and flight plans without changing them. The process is visualized in the activity diagram 3.7.

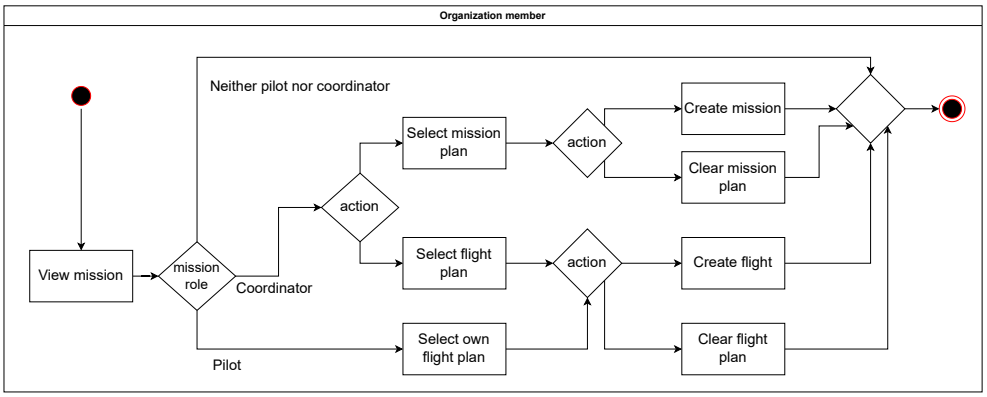

Figure 3.7: Edit Mission Activity Diagram

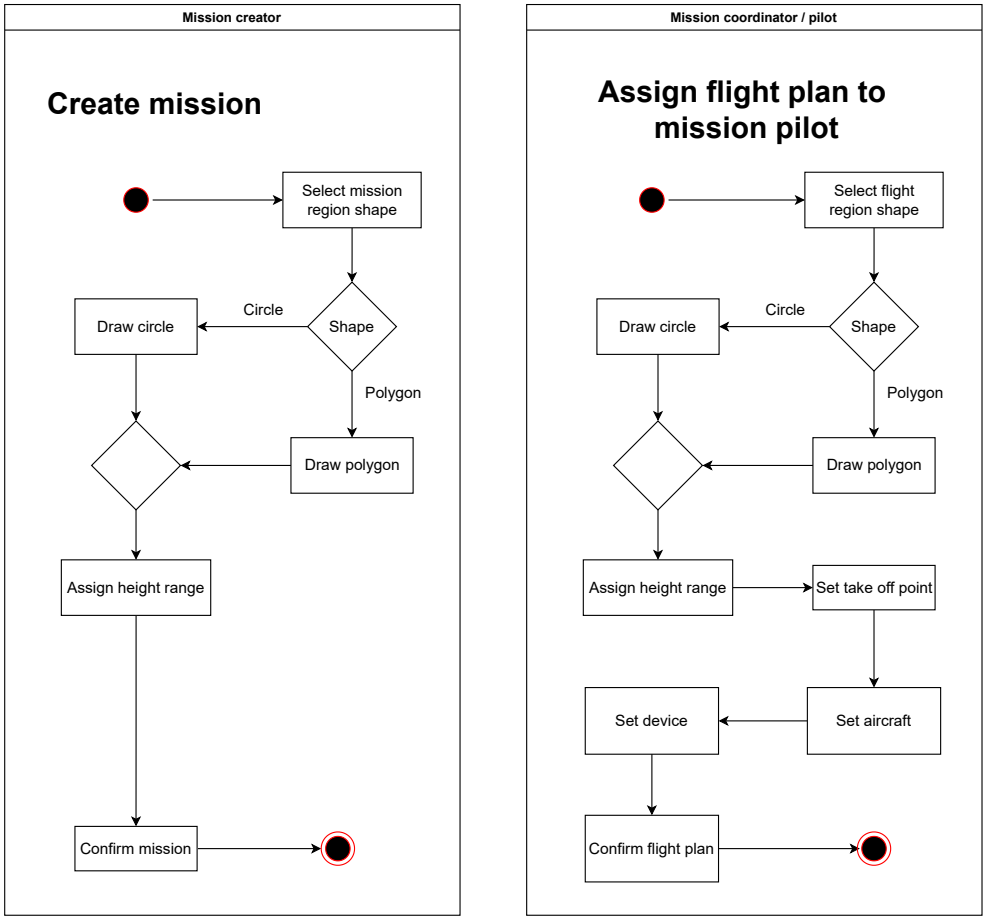

Figure 3.8: Create Mission Activity Diagram

3.2 Organization Module Design

Software architecture and design in React is a very challenging subject. React is a library and does not force developers to write applications in a certain way as frameworks would normally do. It offers freedom that comes at a cost. The design of the module is crafted to be clean, scalable, intuitive, readable, and follows the SOLID principles, explained by Robert C. Martin [36]:

SRP stands for The Single Responsibility Principle, meaning that ”each software module has one, and only one, reason to change”. Since the objective of the thesis is to develop a module, it is possible to further apply this principle onto the files: each source file should have only one responsibility. The SRP was fulfilled as our module is composed from

independent components and each of the components is responsible for only one thing.

OCP or The Open-Closed Principle, which says that the module should be crafted in a way that the behavior of the system should be changed by adding new code, rather than changing the existing code. Simple extensions to the module should not require massive changes in the code that was already written. Our design follows this principle by making sure that the lowest-level UI components are only dependent on the data that they receive, and not on the business logic that manipulates the data.

LSP stands for The Liskov Substitution principle and suggests "building software systems from interchangeable parts, those parts must adhere to a contract that allows those parts to be substituted one for another". In the context of our module it means that components should be interchangeble, which is fulfilled as long as their contract (the props they receive) is respected.

ISP which means The Interface Segregation Principle, advises the developers that they should not depend on things they do not use. The data passed between the components in our module is minimalized.

DIP or The Dependency Inversion Principle, which says that the source code dependencies should refer only to abstractions and not actual implementations. In the context of React, that means that the complex logic should not be implemented directly in the component but imported. Since the module makes use of imported hooks that encapsulate the business logic into small and reusable components, the principle was adhered to.

3.3 Organization Component Design

The web application is split into pages that compose it: Account, Aircraft, Devices, and Flights. Each of these pages has its own:

- types or models representing the data,
- repository that servers as a layer between the container's business logic and API that sends REST HTTP requests to back end and other services,
- hooks handling complex business logic,
- components rendering the UI on the web.

The module will be designed in a similar fashion. It seems very natural to follow this architecture and add a new block called Organization that would be responsible for fleet management. The image 3.9 shows a simple overview of the components that compose the web application.

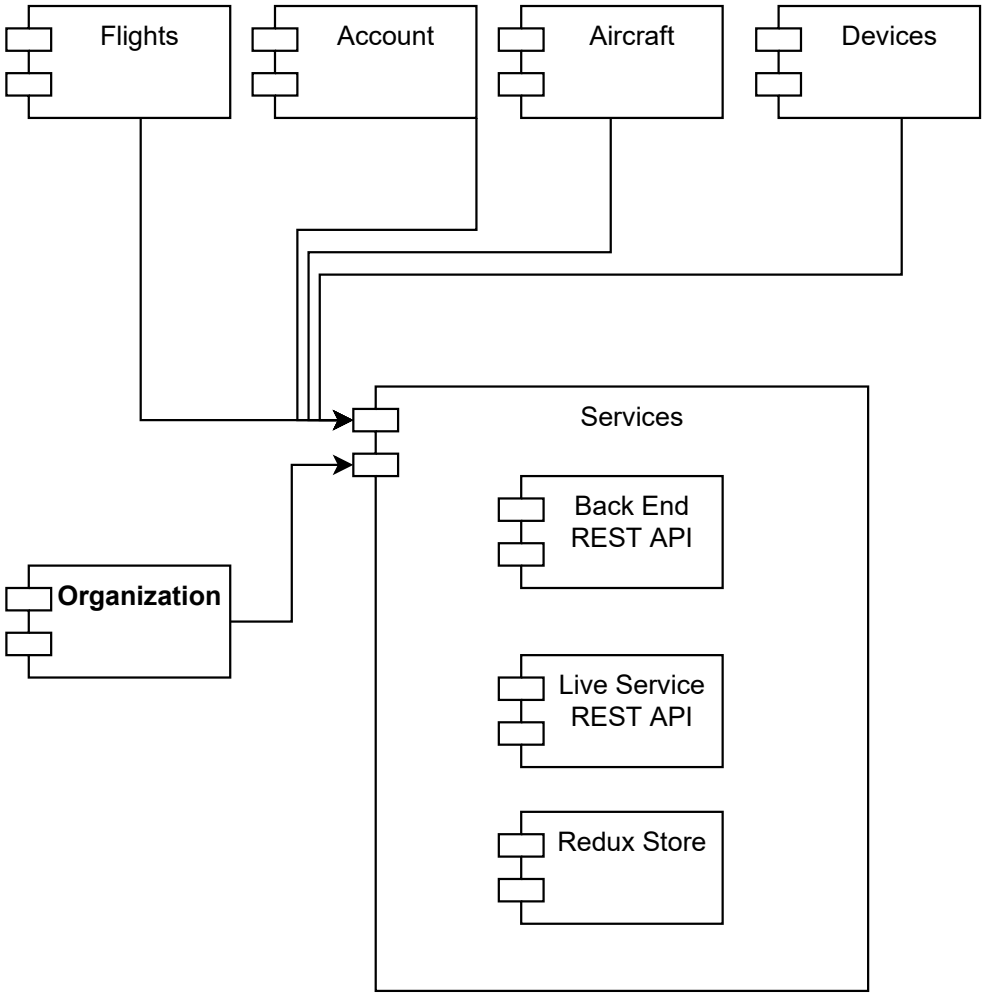

Figure 3.9: Dronetag Front End Component Graph

It is easier to understand the module design after getting familiar with these terms:

Page is a top-level component that renders static elements that should not change and wraps the containers that further divide the application into logical blocks. Static elements usually include the navigation bar, header, and footer.

Container is usually a child of a page or another container. It handles asynchronous calls to other services, business logic, and renders other containers or presentational components.

Presentational component should accept all the data it needs as props and render the UI.

3.3.1 Pages

Organization module is composed of 4 pages: Organization Page, Create Organization Page, Create Mission Page, and Edit Mission Page. These pages serve as wrappers around the corresponding containers, responsible for showing the navigation side bar and styling the page. Each page is responsible for:

- Organization Page renders a container depending on a tab that the user is currently on. There are multiple tabs: Members (default), Teams, Aircraft, Devices, Missions, and Settings,
- Create Organization Page renders Organization Create Container,
- Create Mission Page renders Mission Planner Stepper Container,
- Edit Mission Page renders Mission Plan Edit Container.

For a visual representation, see diagram 3.10. Note that it is not an activity diagram, just a visual representation.

3.3.2 Containers

In this section, we will analyze the containers that each page renders. The containers have a simple lifecycle:

1. Define the local state of the component and retrieve data from other services (hooks),
2. Define functions to handle business logic and local state management (handlers),
3. Handle the loading and error status of asynchronous requests (conditions),
4. Render other containers or presentational components and pass them needed data (return).

Each container retrieves data either from the backend service or from the Redux store using selectors.

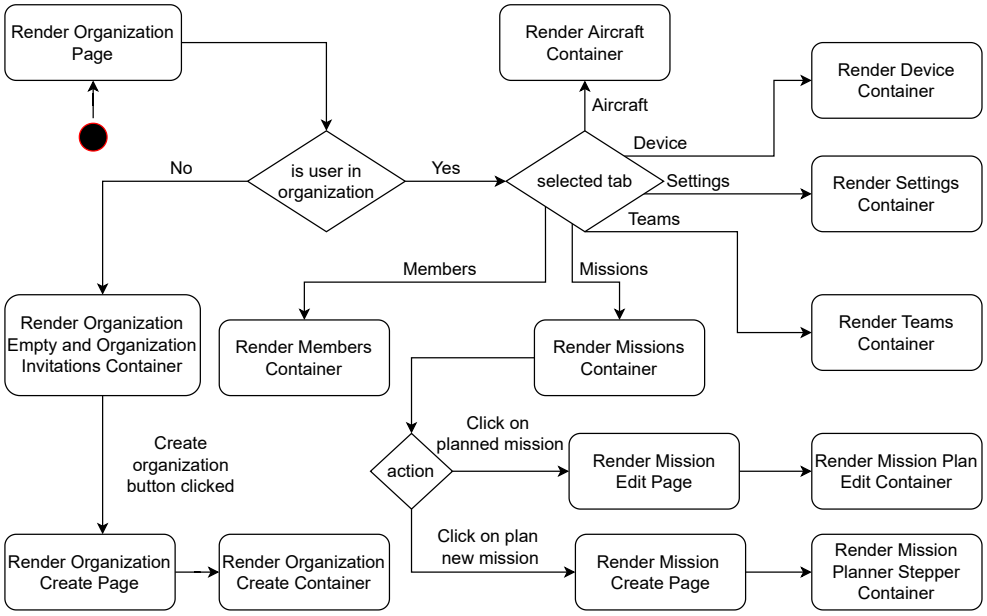

Figure 3.10: Dronetag Page Diagram

Organization Create Container

This container is responsible for rendering a form with the organization name and description. It also handles the form submission logic and call to create an organization.

Organization Members Container

Organization Members Container retrieves organization members data from the back end and holds information about dialogues and pagination in its local state. It renders Member List component and a button to invite users to the organization. It also checks if current user is the organization owner and passes the information to the Member List component.

Organization Aircraft Container

Organization Aircraft Container recovers aircraft data from back end. It renders Aircraft List component and handles pagination. It also renders a button that navigates the user to his own aircraft.

Organization Devices Container

Organization Devices Container works analogously to the Aircraft Container described above.

Organization Settings Container

Organization Settings Container retrieves details about the organization from the back end and checks if the current user is the owner of the organization. It renders an edit form to the organization owner and organization details to other organization members. It will also render a button, either a leave organization button to regular members or a delete organization button to the organization owner.

Organization Missions Container

Organization Missions Container gets mission data from the back end, renders Mission List component, handles the pagination, and renders a button to plan a new mission. After clicking on a mission, the user will be redirected to the Edit Mission Page. After clicking on a button to plan a new mission, the user will be redirected to Create Mission Page.

Mission Planner Stepper Container

Mission Planner Stepper Container handles the stepping mechanism of the Mission Planner and renders the Mission Plan Step One Container or the Mission Plan Step Two Container, depending on the current step.

Mission Plan Step One Container

Mission Plan Step One Container retrieves the organization members data from the back end and renders a form that the user needs to fill to create a Mission Draft, which is an object that holds the necessary static information needed to create an actual mission: name, start date, end date, mission members and their roles corresponding to the mission.

Mission Plan Step Two Container

Mission Plan Step Two Container handles the back end calls and business logic behind the actual mission planning and pilot flight planning. It creates the initial Mission Plan and Flight Plans from the Mission Draft and holds them in a local state.

It then retrieves aircraft and device data from the back end and passes them down to the components responsible for selecting the aircraft and device.

It handles mission or flight plan submit and clear logic. When a plan is submitted, an HTTP POST request is sent to the back end and the mission plan or flight plan is then persisted.

It renders two important components: Mission Pilots Accordion and Mission Map. These components will be explained in detail in the chapter devoted to implementation.

Mission Plan Edit Container

Mission Plan Edit Container works similarly to the Mission Plan Step Two Container, but the initial stage is different, since we need to recreate the flight and mission plans.

It gets information about the existing mission and its mission members in its props. From this information, the container is able to recreate the Mission Draft and fetch the existing Flights from the back end.

The container then creates the mission plan and flight plan objects out of the mission and flight plan information retrieved from back end, and stores these recreated mission and flight plans in the local state of the container.

From this point onward, the logic is analogous to the Mission Plan Step Two Container.

Chapter 4

Implementation

In this chapter, the more advanced implementation details and implementation challenges of the module will be explained: used React libraries, data synchronization, UI rendering, form and dialogue handling, memoization, and refactoring. Then we will dig deeper into the actual components and algorithms behind mission planning.

4.1 Data Synchronization

Data synchronization includes fetching, caching, synchronizing, and updating the server state. Our module has four layers of abstraction over data fetching:

- Queries and Mutations – React Query Hooks that handle backend calls in containers and provide the data and status information,
- Repository – middle layer responsible for conversions between camel case and snake case, parsing the response headers, and data mapping,
- REST API – collection of requests to REST Client,
- REST Client – the last layer responsible for actual backend calls using the global fetch method from JavaScript Fetch API [37].

React Query

React Query [38] is a powerful data fetching library for React. It has three core concepts:

Queries are React Query Hooks used for data fetching. Each query needs a Promise-based method and a unique key that are used to refetch the data and cache them in the Query Client. The query returns an object that has 4 mutually exclusive states: *isLoading*, *isError*, *isSuccess*, and *isIdle*. Depending on the state, the object holds the data or error information.

Mutations are React Query Hooks used to create or update the server data and perform side effects. Mutations also return an object with the same states as queries and provide us with the data or error information. Very important feature of mutations is the lifecycle options: *onMutate*, *onError*, *onSuccess*, and *onSettled*. These functions handle the asynchronous logic and allow us to execute our own logic that depends on the current state of backend calls.

Query invalidation provides methods used to handle cached data in the Query Client. A common use case is marking cached query results that depend on other data as stale when the other data changes.

React Query library has out-of-the-box solutions for many advanced and complex concepts such as parallel data fetching, dependent queries, infinite queries, window focus refetching, or query batching. It also has first-class support for TypeScript, provides powerful debugging web tools, and requires minimal configuration.

4.2 Memoization

Memoization is an optimization technique used in programming to improve performance by storing information that is expensive to calculate. Whenever the same information is requested, it is only returned from the store instead of being calculated all over again.

In the context of a React application, it is used to optimize performance of the application by preventing unnecessary rendering. Whenever a parent component is changed, there are difference checks run on every child even if there were no changes to their props. We can prevent this difference check by using memoization. It is realized by making use of three functions:

React.memo is a higher order component that wraps the component that is memoized. By default, it performs a shallow equality check on props, but it is possible to create a custom equality check function and pass it to React.memo as an argument.

React.useMemo is a hook that returns a memoized value. It takes a factory function and a dependency array, recalculating the value only if the values in the dependency array change.

React.useCallback is a hook that returns a memoized callback. It works similarly to React.useMemo and is used to help memoized components that accept functions as props.

In the module, memoization had to be implemented to improve Mission Planner performance.

4.3 Refactoring

Refactoring plays a critical role in the software implementation and there was a lot of work spent on refactoring the module. Fowler's [39] definition of refactoring states the following:

> *"Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure. It is a disciplined way to clean up code that minimizes the chances of introducing bugs. In essence when you refactor you are improving the design of the code after it has been written."*

In React, the components tend to get heavy very quickly. Heavy components are hard to read, break the SOLID principles and they tend to slow the module down, because it gets challenging to optimize the components properly.

To demonstrate how refactoring helped make the code of the module cleaner, let us look at Mission Pilots Accordion component. At first, the component was directly responsible for rendering the mission accordion and all the accordions for mission pilots. The logic handling the selection of the aircraft, device and height range was implemented in the Mission Plan Step Two Container. While the application worked, it had a critical impact on:

- readability of the code – Mission Plan Step Two Container kept growing in size with each addition to the Mission Pilots Accordion,
- performance – a change in a single flight plan caused re-rendering of all the accordions instead of re-rendering a single isolated component.

Similar changes were made to Mission Map component, which originally rendered all of its drawn regions on the map instead of delegating the responsibilities to specific and isolated components. Refactoring is a very important part of writing a maintainable, clean, readable and performant code.

4.4 UI Components

The UI design of the web application is very important to users. User interface should be clean, intuitive and look modern. The module makes use of a powerful UI suite, Material UI.

Material UI

Material UI (MUI) [40] library provides numerous styled React components used to present data to the user. The library is mostly used for styling and

overriding the provided components by defining custom styles and using the *className* prop of each MUI component.

In the module, the CSS styles are specified in an object that we pass to *createStyles* callback, which is called inside of the *makeStyles* function. Each MUI component is then assigned its style by setting the *className* prop to the desired object that holds the style information.

4.5 Forms

Forms are present in almost every user-oriented web application. They provide a way to catch user input and submit it elsewhere. Forms need a way to validate the fields, show error messages, and submit themselves. In React, this can become complicated and verbose very quickly. The module makes use of Formik library to mitigate these issues.

Formik

Formik [41] is a React library that provides an easy way to build, validate and submit forms with minimal API. It also has first-class support for TypeScript and MUI, the UI library used in the module.

Formik form accepts initial values (usually the data model), validation schema, and a callback to be executed when submitting the form. Each form field is defined and given an ID corresponding to the object keys in the initial values. Other functionality like updating the values can be left on Formik to handle, but it is possible to use the provided methods to customize the behavior.

4.6 Dialogues

Dialogues are handled by the MUI library. Each dialogue accepts information about its state (open/closed) and a callback that handles the confirm logic as props. Dialogues usually render a ”go back” button that closes the dialogue, and a submit button that fires the callback passed to it.

The module uses eight dialogues. The dialogs are used for safety and informational reasons. They prevent users from accidentally performing destructive actions like deleting an organization or a mission, and serve as an informative overview before creating a mission or a flight.

4.7 Map

The module uses react-map-gl [42] library to load an interactive map. It is based on Mapbox GL JS [43], which is a JavaScript library for building and customizing an interactive map using Mapbox's modern mapping technology.

Map is a React component that accepts props used to control the map such as mouse events, latitude, longitude, or zoom.

It is possible to ”draw” onto the map by rendering components as children of the Map component. Any React component can be used, but it is recommended to use the components from react-map-gl library. The most frequently used components are:

- Marker – component that accepts longtitude, latitude and other useful props such as drag and click events, offsets, or anchor, and serves as a wrapper around a component or icon,
- Layer – component that creates a map layer. Layer requires a style; we can choose from multiple Mapbox layer styles like fill, line, symbol, and many others,
- Source – component that creates a map source from multiple source objects such as a vector or geojson. Can contain Layers as its children.

4.8 Mission Plan Model

Mission Plan is a type that represents a mission plan. It encapsulates the following properties:

- Shape – defines the geometric shape of the plan, can be either NotSelected, Polygon, or Circle,
- HeightRange – an array of two integers representing the lower and upper altitude of the plan,
- PolygonRegionInfo – an object that holds information concerning Polygon Region,
- CircularRegionInfo – an object that holds information concerning Circular Region,
- Color – string that represents color of the plan,
- Confirmed – a boolean that marks the plan as confirmed.

Mission plan and flight plans are stored in the local state of Mission Plan Step Two and Mission Plan Edit containers. These plans serve as the source of truth, and the containers pass them down the component tree with functions used to mutate the plans. Whenever a flight region is dragged on the map, the corresponding flight plan gets updated and every components that accepts the flight plan as props gets re-rendered.

4.9 Mission Planning Components

In this section, the components responsible for mission planning will be explained in more detail.

Mission Pilots Accordion

Mission Pilots Accordion is a component responsible for rendering Mission Accordion and Pilot Accordion components. It holds information about the expanded accordion in its local state, and expanding an accordion changes the selected plan in the Mission Plan Step Two Container or Mission Plan Edit Container. It also defines functions used to select the shape of a plan region, assign height range to a plan, or select an aircraft or a device to a flight plan. It passes these functions to its children, Mission Accordion and Pilot Accordion components.

To see a visual representation of the component, see image 4.1.

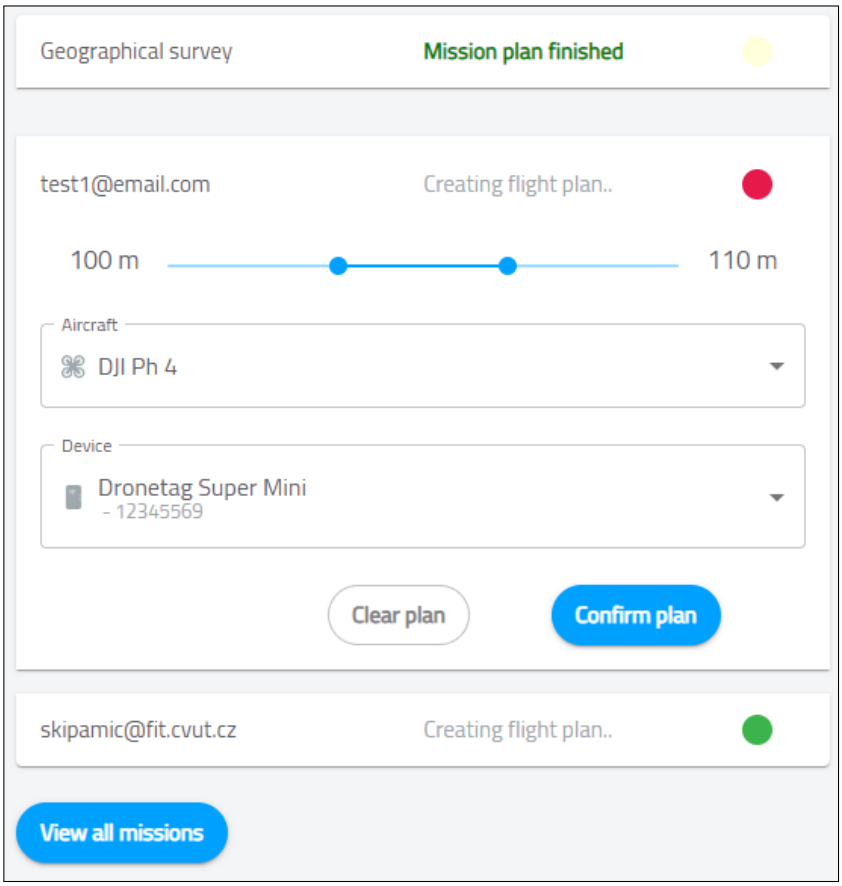

Figure 4.1: Mission Pilots Accordion Component

Mission Accordion

Mission Accordion renders Shape Select and Height Select components, buttons to clear or confirm the mission plan, and general information about the state of the plan. Accordion is disabled for non-coordinators during mission plan editing, as clearing the plan would remove the mission and its flights.

Pilot Accordion

Pilot Accordion is built similarly to Mission Accordion, but it also renders Aircraft Select and Device Select components.

Mission Map

Mission Map is a component that renders the React Map component from the react-map-gl library. It holds viewport in its local state, which defines the position on the map. It defines functions to handle map clicking, viewport changes, and mouse movement. It renders Polygon Region, Circular Region, and Takeoff Marker components.

For the visual representation of the component, see image 4.2.

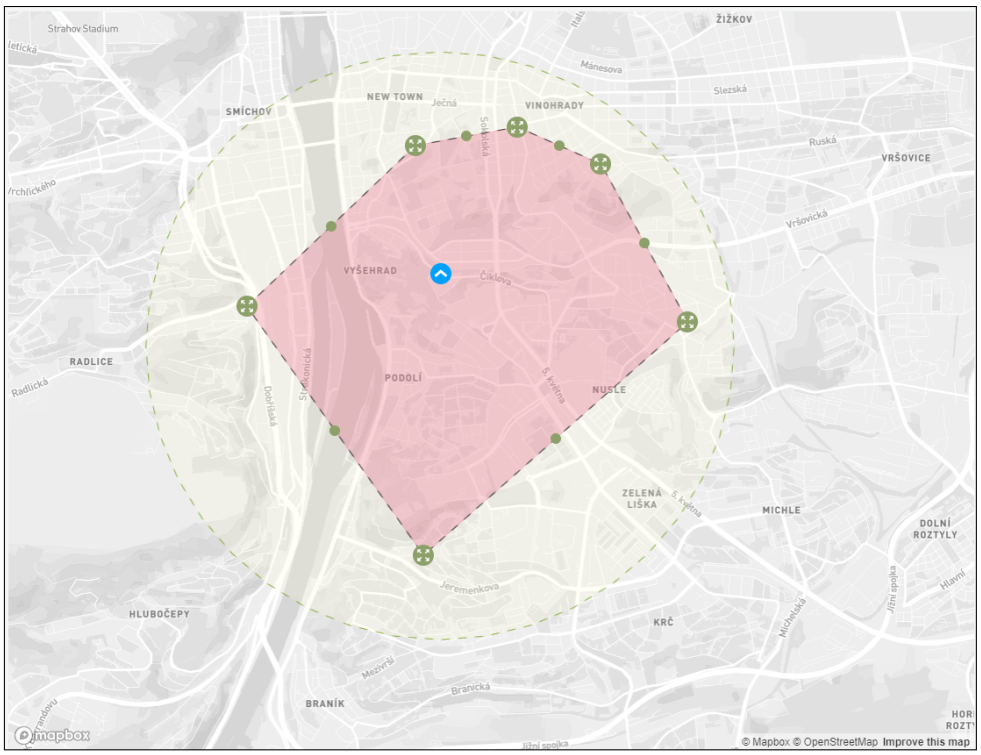

Figure 4.2: Mission Map Component

Polygon Region

Polygon Region renders Source components calculated from the plan's polygon region information and Layer components that graphically represent these Source components. The component defines functions that are used to manipulate Region Points and Region Midpoints components. It also renders drawing instructions that guide the user while drawing the plan.

Region Points

Region Points component renders a Marker for each polygon region point.

Circular Region

Circular Region is similar to Polygon Region, but renders a Circular Region Center Marker and Circle Resize Point which are used to move or resize the Circular Region.

Overview

The components described in the sections above are children of the Mission Plan Step Two Container or Mission Plan Edit Container, communicate with each other and mutate the mission plan and flight plans stored in the local state of the corresponding containers. To see the full picture of the Mission Plan Step Two Container, refer to image 4.3.

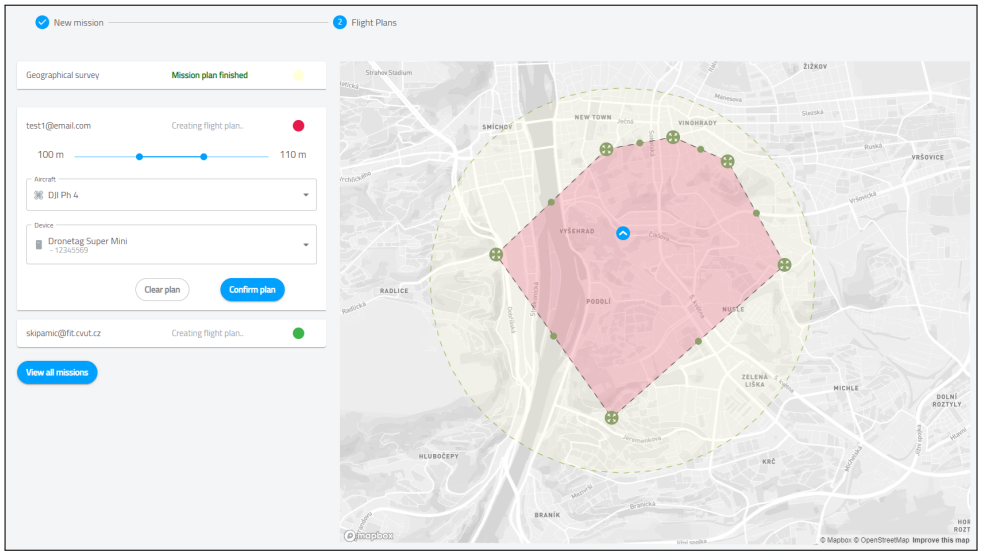

Figure 4.3: Mission Plan Step Two component

Chapter 5

Testing and Feedback Evaluation

The last chapter will focus on the last objectives of the thesis: testing the application with real pilots, evaluating the results, and suggesting future improvements. In the following sections, each of these topics will be further elaborated on.

5.1 Testing

Testing plays a crucial role in the software development. The definition from The Art of Software Testing [44] states:

> *"Software testing is a process, or a series of processes, designed to make sure computer code does what it was designed to do and, conversely, that it does not do anything unintended. Software should be predictable and consistent, presenting no surprises to users."*

Since the testing process of the web application was not previously defined, it was decided that testing the code in a form of unit or integration tests would be out of scope of the thesis. To ensure that the module works correctly, the application was tested on pilots familiar with the Dronetag, and users that had never seen the application before.

5.1.1 User Testing Process

The process of user testing was designed to be as close as possible to a real user experience. The users were instructed to perform specific tasks without any external guidance, which made sure the use cases were tested, as well as the user interface, and the application as a whole.

5.1.2 Test Scenarios

Specific test scenarios will be described and labeled in this section.

TC01 Organization Creation

The first test scenario tests FR01 2.5. The user who is not a member of any organization is told:

> *"You are a leader of a drone fleet in your company and you want to create an organization for your department. The name of the organization will be 'Dronefleet Organization' and the description will state 'Organization that plans complex drone operations'."*

This test verifies that:

- User is able to find the organization section from the main page,
- User is able to create an organization with a specific name and description.

TC02 Organization Management

The second test scenario tests FR01 2.5. The user who is an organization owner is told:

> *"You want to rename the newly created organization to 'Dronefleet Company' and change the description to 'Company that plans complex drone mission'."*

This test makes sure that:

- User is able to find the organization settings,
- User is able to edit the organization.

TC03 Organization Member Management 1

This scenario tests FR03 2.5. The user is told:

> *"Your task is to invite your two colleagues, John and David with emails 'john@email.com' and 'david@email.com' to your organization. After that, sign out, sign in again as John and join the organization."*

This test verifies the following:

- User is able to invite other users to the organization by their email,
- User is able to join an organization he was invited to.

TC04 Organization Member Management 2

This scenario tests FR03 2.5. The user is told:

> *"While signed is as John, try to remove any user from the organization. Change the organization name to 'Corrupted company'. After that, try to delete the organization. Then sign out and sign in back to your account."*

This test verifies the following:

- Organization member that is not the owner of the organization cannot remove any organization members,
- Organization member that is not the owner of the organization cannot edit or delete the organization.

TC05 Organization Asset Management

This test scenario tests FR02 2.5 and states:

> *"You have two drones and a Dronetag device saved on your account. Check your organization aircraft and transfer these two drones to your organization. Then check your organization devices and transfer the device to your organization. After that, transfer one of the aircraft back to your personal account."*

This test ensures that:

- User is able to view organization aircraft,
- User is able to transfer his aircraft to organization,
- User is able to transfer organization aircraft to his account,
- User is able to view organization devices,
- User is able to transfer his device to organization,
- User knows where to look for organization assets and personal assets.

TC06 Mission Planning

In this test scenario, FR05 2.5 is put to the test. The user is told:

> *"Your task is to plan a mission named 'Geographical survey'. The mission will take place tomorrow from 14:00 to 15:30. There will be three people involved in this mission: You, John, and Dave. You and John will be pilots, Dave will not be flying but he will be a coordinator. John will not be a coordinator. The mission will take place at Stromovka park, so you will create a polygonic mission plan over the entire Stromovka park. The altitude will be 80 to 150 meters."*

This test case verifies that:

- User knows how to start planning a mission,
- User is able to specify the mission name and planned date,
- User is able to add organization members to the mission,
- User is able to set the roles of the mission members,
- User is able to create a mission plan region over a specified area,
- User is able to set mission altitude.

TC07 Mission Flight Planning

This test scenario takes place immediately after TC6 and also tests FR05 2.5. The user is told that:

> *"You will be flying in the eastern half of the mission plan and you will be taking off in the middle of the eastern half. You should fly in range of 90 to 120 meters. You will be flying with aircraft named 'DJI Ph4' and Dronetag device named 'Dronetag Mini 1'."*

This test covers the following:

- Mission creator is able to create his flight region within a specified mission plan region,
- Mission creator is able to place a take-off marker in a specified location,
- Mission creator is able to set the height range of the flight plan to be different from the mission plan's height range,
- Mission creator is able to find and select the correct aircraft for his flight,
- Mission creator is able to find and select the correct device for his flight,
- Mission creator is able to create a valid flight plan.

TC08 Planned Mission Edit Pilot

The following test scenario tests FR05 2.5 and the user gets the following instructions:

> *"Sign in as John and find a planned mission called 'Geographical Survey'. Try to clear the mission plan and edit the other flight plans. Then create a flight plan region that covers the western part of the mission plan region. It should not intersect with any other flight plans. You will be taking off from the center of the flight plan region and you will be flying at altitude of 100 to 110 meters. You can select any aircraft or device that you want to fly with."*

This test ensures that:

- Mission pilot is able to view a planned mission,
- Mission pilot knows how to edit a planned mission,
- Mission pilot cannot edit or delete the mission plan or any other flight plans except for his own,
- Mission pilot is able to create the specified flight plan.

TC09 Planned Mission Edit Not Coordinator Or Pilot

This test scenario tests FR05 2.5 and the user receives the following instructions:

> *"Sign in as Dave and view a planned mission called 'Geographical Survey'. Try to clear the mission plan and every flight plan."*

The test scenario ensures that:

- Organization member that is neither a pilot nor a mission coordinator can view a mission,
- Organization member that is neither a pilot nor a mission coordinator cannot edit the mission plan or any of the flight plans.

TC10 Flight Plan Region Not in Mission Plan Region

The following test scenario tests FR05 2.5. The user is told:

> *"Find a planned mission called 'Boundaries Test'. Create an arbitrary flight plan region outside of the mission plan, select any aircraft, device, and set the take-off point inside the flight plan. Try to confirm the flight plan. Then move the flight plan region so that it is inside the mission plan. Confirm the flight plan."*

This test scenario verifies that:

- Mission pilot cannot submit a flight plan that is outside of the mission plan region,
- Mission pilot is able to submit a correct flight plan.

5.2 Testing Report

Testing with users was realized on 27 April 2022 and 10 May 2022 on four test subjects. Two of them were not familiar with Dronetag. The testing was a success and revealed that the module was designed and implemented well. The test subjects reported the following inconveniences that they encountered during the testing:

- When drawing a polygon, it is not immediately apparent how to finish drawing the plan,
- Sign out button should be placed directly on the navigation side bar instead and not in User Settings,
- There is no way to add additional mission members after the mission has been planned,
- Moving the circular region could be smoother,
- The instruction to set take-off marker is hard to spot.

The suggested solutions to some of these problems can be found in the next chapter.

5.3 Future Improvements

The application is still a prototype. There are many things that could be worked on and functionalities that should be added to the module. The following section will describe the current problems with the application and suggest solutions to these problems.

Implement Teams Page

Problem: Teams page was designed, but there was not enough time to implement it.

Suggested solution: Implement the Teams page according to the graphic prototype with functionalities defined in Functional Requirements section.

Ongoing and Past Missions View

Problem: The Missions section of the module does not differentiate between planned, ongoing, and past missions. Clicking on a planned mission takes the user to the Mission Edit page, and clicking on an ongoing or past mission does not do anything.

Suggested solution: Implement the Mission Detail page according to the graphic prototype that would present the most important metrics of a past or ongoing mission. Add a way to filter planned, ongoing and past missions.

Switching Between Organization and Personal Account

Problem: The Organization module resides in the navigation sidebar and navigation between containers is handled by clicking on tabs.

Suggested solution: Enable the user to switch between organization and personal account in the navigation sidebar. The navigation sidebar would now handle the navigation between the pages of the organization module

Add and Remove Mission Member Functionality

Problem: The user adds members to the mission in the first step of mission planning and cannot add members to the mission after proceeding to the next step.

Suggested solution: Add a button that would take the user back to the first step of the mission. The user would be able to edit the mission information including the mission members.

Conclusion

The objectives of the thesis were to analyze the market for existing solutions, define the functionalities of the planned module, implement the module, test the finished module with real pilots, and suggest its future improvements.

I managed to analyze the market and research existing drone fleet management solutions. I evaluated the platforms and listed their positives and negatives. It gave me all the necessary information and experience to come up with my own solution to this problem.

After many iterations, I defined the final functionalities of the planned module for the Dronetag application. The analysis of other drone fleet management applications revealed which features I should put the most focus on and functionalities that were not that important or interesting to the user. The functional requirements had served me as a guideline while I was designing the graphical prototype in Adobe XD.

The graphical prototype has proven to be the perfect foundation for the implementation of the module. It made the actual implementation much easier because I knew what the UI should look like at all times. It also helped me identify edge cases and possible problems in the application.

It took me a long time to get familiar with the existing codebase, but it was a great experience. I went into the depths of code that I had not written. Once I knew the codebase inside and out, I was able to come up with the architecture and high-level design of the module. After that I could start implementing my precisely crafted solution.

The implementation was the heart of this thesis, and with over 8000 lines of code, I managed to successfully implement the module and integrate it into the existing Dronetag application. After many refactorings and changes, I delivered a clean code following the best practices of JavaScript, TypeScript, and especially the React library. I used a simple, intuitive, and easily extensible architecture.

Testing the application module with real pilots convinced me that my solution fulfills the user needs for drone fleet management, but it is not perfect.

Software is all about refactoring and changes in requirements according to feedback provided by users. I thoroughly evaluated user feedback and suggested possible improvements to the module that could be done in the future.

In the end, it is important to note that the module is still a prototype and there was not enough time to implement several designed functionalities. There are many possible improvements that could be implemented, but that would be out of scope of this bachelor's thesis.

Bibliography

1. AIRHUB. *Airhub: Drone Operations Center* [online]. AirHub, 2021 [visited on 2022-04-24]. Available from: `https://www.airhub.app/drone-operation-center`.
2. AIRHUB. *AirHub App* [online]. AirHub, 2021 [visited on 2022-05-11]. Available from: `https://dashboard.airhub.app/`.
3. DJI. *DJI Introduces FlightHub Software To Help Enterprises Efficiently Manage Their Drone Operations* [online]. DJI, 2017 [visited on 2022-04-24]. Available from: `https://www.dji.com/newsroom/news/dji-introduces-flighthub-software-to-help-enterprises-efficiently-manage-their-drone-operations`.
4. DJI. *DJI FlightHub Download Center* [online]. DJI, 2022 [visited on 2022-04-24]. Available from: `https://www.dji.com/cz/downloads/products/flighthub`.
5. DJI. *DJI FlightHub Enterprise User Guide* [online]. DJI, 2017 [visited on 2022-05-11]. Available from: `https://dl.djicdn.com/downloads/FlightHub/20190308/FlightHub_Enterprise_User_Guide_v1.0_EN.pdf`.
6. DRONETAG S.R.O. *Dronetag - Tvoříme bezpečný svět dronů* [online]. Dronetag s.r.o., 2020 [visited on 2022-04-24]. Available from: `https://dronetag.cz/`.
7. DRONETAG S.R.O. *Dronetag* [online]. Dronetag s.r.o., 2022 [visited on 2022-05-11]. Available from: `https://dronetag.app/`.
8. THE POSTGRESQL GLOBAL DEVELOPMENT GROUP [online]. The PostgreSQL Global Development Group, 2022 [visited on 2022-05-09]. Available from: `https://www.postgresql.org/`.
9. PYTHON SOFTWARE FOUNDATION. *Welcome to Python.org*. Python Software Foundation, 2022. Available also from: `https://www.python.org/`.

10. FOUNDATION, Django Software [online]. Django Software Foundation, 2022 [visited on 2022-05-11]. Available from: https://www.djangoproject.com/.

11. ENCODE OSS LTD. *Django REST framework* [online]. Encode OSS Ltd., 2022 [visited on 2022-05-11]. Available from: https://www.django-rest-framework.org/.

12. SOLEM, Ask. *Celery - Distributed Task Queue*. Ask Solem, 2021. Available also from: https://docs.celeryq.dev/en/stable/.

13. DART COMMUNITY. *Dart programming language* [online]. Dart Community, 2022 [visited on 2022-05-11]. Available from: https://dart.dev/.

14. FLUTTER. *Flutter - Build apps for any screen* [online]. Flutter, [n.d.] [visited on 2022-05-11]. Available from: https://flutter.dev/.

15. THE BLOC COMMUNITY. *Bloc, a predictable state management library for Dart* [online]. the Bloc Community, 2022 [visited on 2022-05-11]. Available from: https://bloclibrary.dev/#/.

16. JAVASCRIPT.COM. *CELEBRATING 25 years of JavaScript* [online]. JavaScript.com, 2022 [visited on 2022-05-11]. Available from: https://www.javascript.com/.

17. ECMA INTERNATIONAL. *Draft ECMA-262 / April 28, 2022* [online]. Ecma International, 2022 [visited on 2022-04-28]. Available from: https://tc39.es/ecma262/.

18. MICROSOFT. *TypeScript: JavaScript With Syntax For Types* [online]. Microsoft, 2022 [visited on 2022-05-09]. Available from: https://www.typescriptlang.org/.

19. META PLATFORMS, INC. *React – a JavaScript library for building user interfaces* [online]. Meta Platforms, Inc., 2022 [visited on 2022-05-09]. Available from: https://reactjs.org/.

20. REFSNES DATA. *React JSX* [online]. Refsnes Data, 2022 [visited on 2022-05-04]. Available from: https://www.w3schools.com/react/react_jsx.asp.

21. REFSNES DATA. *JavaScript HTML DOM* [online]. Refsnes Data, 2022 [visited on 2022-05-04]. Available from: https://www.w3schools.com/js/js_htmldom.asp.

22. SHETH, Kishan. *What is Virtual DOM?* DEV Community, 2021. Available also from: https://dev.to/koolkishan/what-is-virtual-dom-how-virtual-dom-works-what-is-reconciliation-what-is-diffing-algorithm-what-makes-react-so-fast-327a.

23. GAMMA, Erich; HELM, Richard, et al. Observer. In: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994. ISBN 0201633612.

24. PROGRAMMING WITH MOSH. *Virtual DOM* [online]. Programming with Mosh, 2015 [visited on 2022-05-11]. Available from: `https://programmingwithmosh.com/react/react-virtual-dom-explained/`.

25. META PLATFORMS, INC. *Components and Props* [online]. Meta Platforms, Inc., 2022 [visited on 2022-05-04]. Available from: `https://reactjs.org/docs/components-and-props.html`.

26. META PLATFORMS, INC. *State and Lifecycle* [online]. Meta Platforms, Inc., 2022 [visited on 2022-05-04]. Available from: `https://reactjs.org/docs/state-and-lifecycle.html`.

27. WOJCIECH MAJ. *React lifecycle methods diagram* [online]. Wojciech Maj, 2019 [visited on 2022-05-11]. Available from: `https://projects.wojtekmaj.pl/react-lifecycle-methods-diagram/`.

28. META PLATFORMS, INC. *Thinking in React* [online]. Meta Platforms, Inc., 2022 [visited on 2022-05-04]. Available from: `https://reactjs.org/docs/thinking-in-react.html`.

29. META PLATFORMS, INC. *Lifting State Up* [online]. Meta Platforms, Inc., 2022 [visited on 2022-05-04]. Available from: `https://reactjs.org/docs/lifting-state-up.html`.

30. META PLATFORMS, INC. *Introducing Hooks* [online]. Meta Platforms, Inc., 2022 [visited on 2022-05-09]. Available from: `https://reactjs.org/docs/hooks-intro.html`.

31. MALAN, Ruth; BREDEMEYER, Dana, et al. Functional Requirements and Use Cases. *Bredemeyer Consulting*. 2001.

32. WIEGERS, Karl Eugene; BEATTY, Joy. Understanding User Requirements. In: *Software Requirements, Third Edition*. Microsoft Press, 2015.

33. CHUNG, Lawrence; NIXON, Brian A., et al. *Non-functional requirements in software engineering*. Springer Science & Business Media, 2012. ISBN 9781461552697.

34. *Adobe XD* [online]. Adobe, 2022 [visited on 2022-05-04]. Available from: `https://www.adobe.com/products/xd.html`.

35. STRIDE XL, S.R.O. *Stride XL* [online]. Stride XL, s.r.o., 2021 [visited on 2022-05-11]. Available from: `https://www.stridexl.com/`.

36. MARTIN, Robert C. Design Principles. In: *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Pearson, 2017, pp. 57–91.

37. KANTOR, Ilya. *Fetch* [online]. Ilya Kantor, 2022 [visited on 2022-05-11]. Available from: `https://javascript.info/fetch`.

38. LINSLEY, Tanner. *Performant and powerful data synchronization for React* [online]. Tanner Linsley, 2020 [visited on 2022-05-04]. Available from: `https://react-query.tanstack.com/`.

39. FOWLER, Martin; BECK, Kent, et al. Preface. In: *Refactoring - Improving the Design of Existing Code*. Addison-Wesley Professional, 1999.

40. MATERIAL UI SAS. *MUI: The React component library you always wanted* [online]. Material UI SAS., 2022 [visited on 2022-05-04]. Available from: `https://mui.com/`.

41. FORMIUM, INC. *Build forms in React, without the tears* [online]. Formium, Inc., 2020 [visited on 2022-05-04]. Available from: `https://formik.org/`.

42. URBAN COMPUTING FOUNDATION. *React-Map-GL* [online]. Urban Computing Foundation, 2020 [visited on 2022-05-04]. Available from: `https://visgl.github.io/react-map-gl/`.

43. MAPBOX. *Mapbox GL JS* [online]. 2014 [visited on 2022-05-04]. Available from: `https://docs.mapbox.com/mapbox-gl-js/guides/`.

44. MYERS, Glenford J.; SANDLER, Corey; BADGETT, Tom. A Self-Assessment Test. In: *The Art of Software Testing*. John Wiley & Sons, 2012.

Appendix A

Graphic Prototype

This chapter will contain the graphic prototype pages that were not shown in the previous chapters.

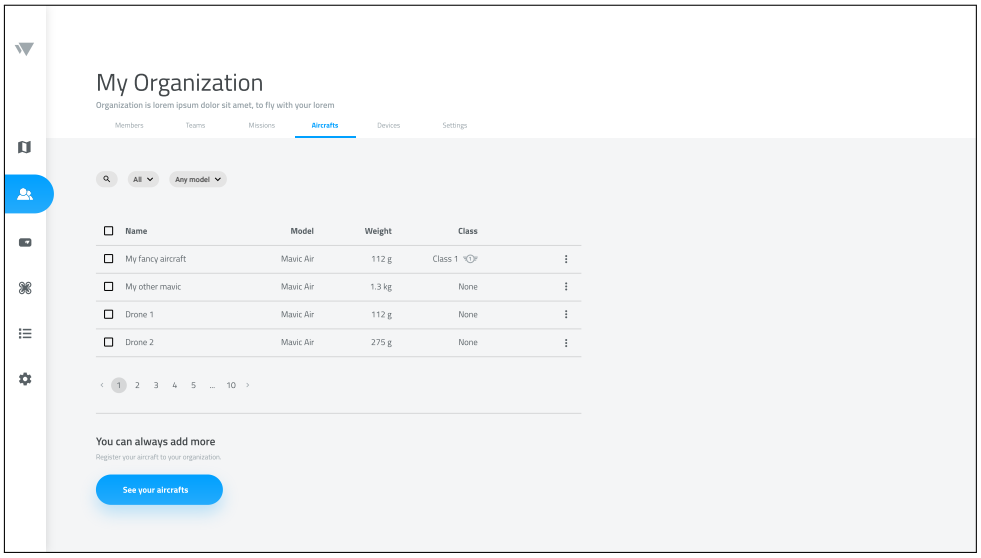

Figure A.1: Organization Aircraft Page

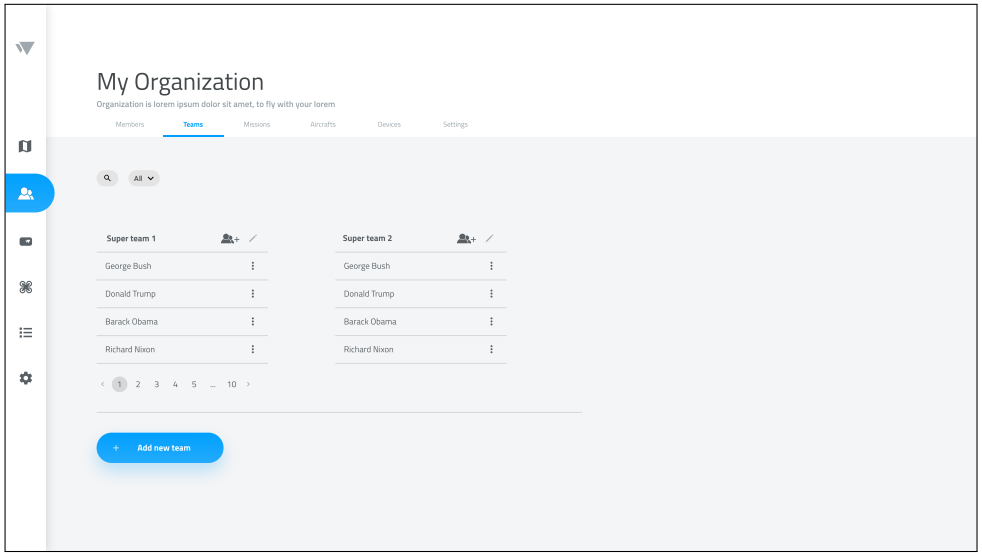

Figure A.2: Organization Teams Page

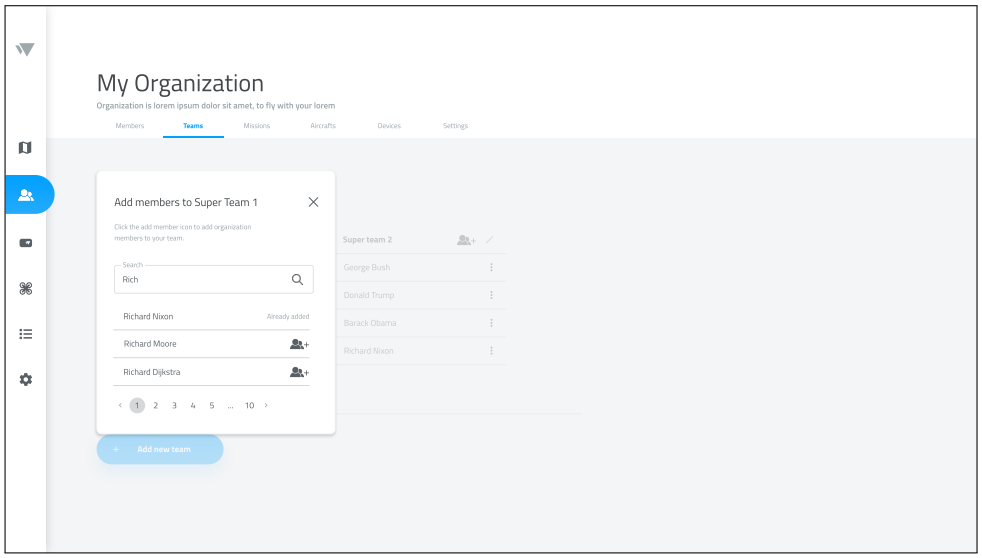

Figure A.3: Organization Teams Add Members Page

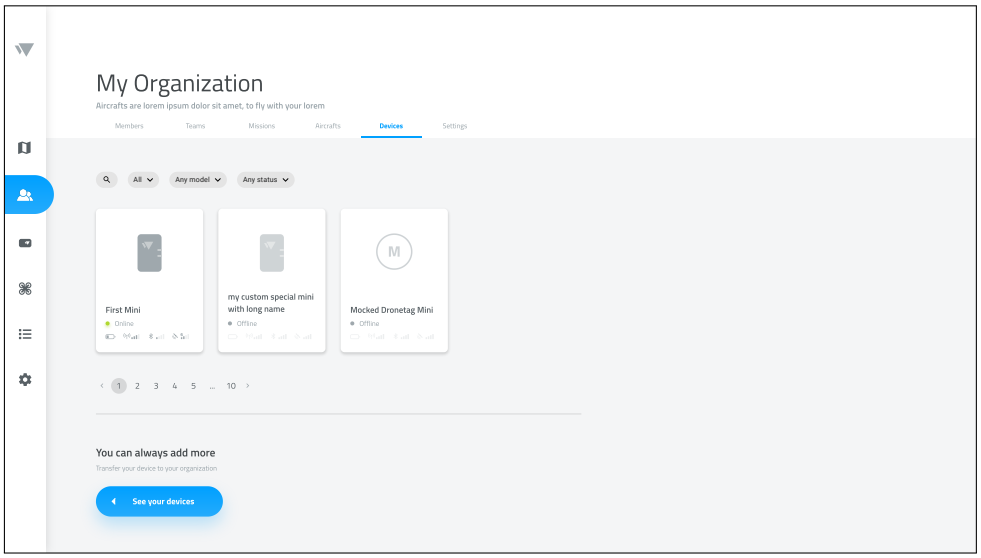

Figure A.4: Organization Devices Page

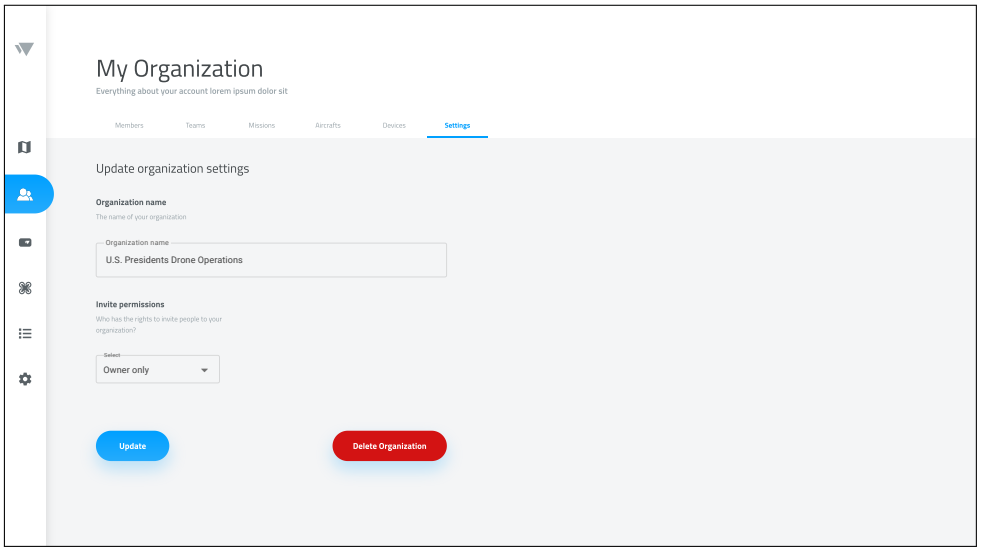

Figure A.5: Organization Settings Page

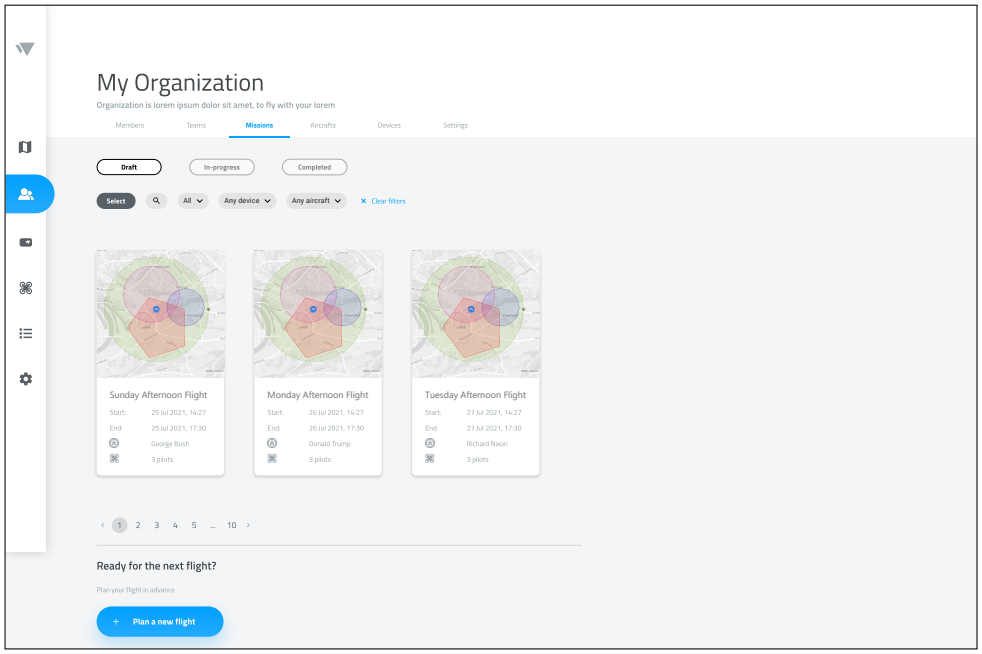

Figure A.6: Organization Missions Page

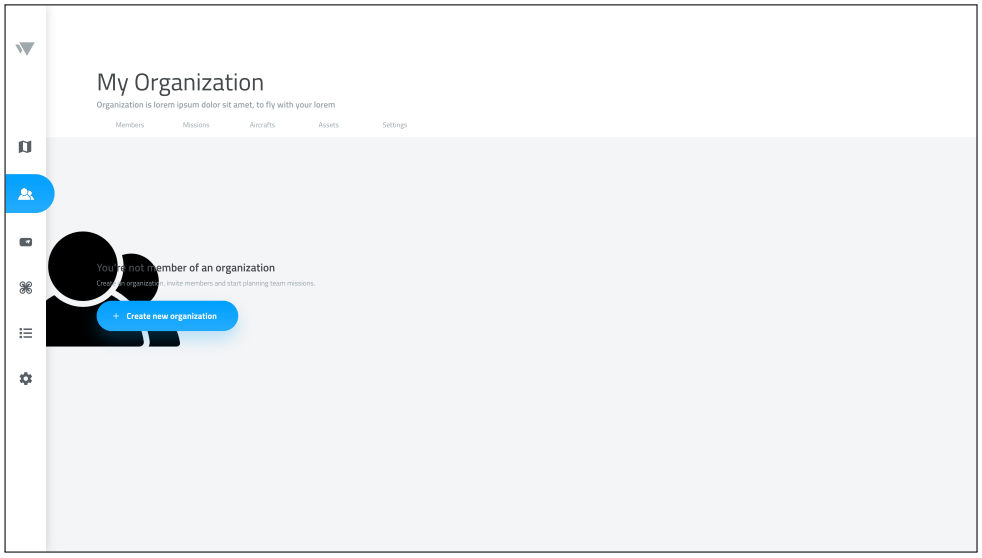

Figure A.7: Organizations Create Page

Appendix B

Acronyms

API Application programming interface.

BVLOS Beyond visual line of sight.

CSS Cascading Style Sheets.

DJI Da-Jiang Innovations.

DOM Document Object Model.

EGNOS European Geostationary Navigation Overlay Service.

GLONASS Global Navigation Satellite System.

GPS Global Positioning System.

HTML HyperText Markup Language.

HTTP Hypertext Transfer Protocol.

JIT Just in Time.

JSX JavaScript XML.

MUI Material UI.

OOP Object Oriented Programming.

REST Representational state transfer.

UI User interface.

UTM Unmanned Traffic Management.

UX User experience.

VLOS Visual line of sight.

Appendix C

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
  - `thesis.pdf` ..... the thesis text in PDF format