BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

WEB APPLICATION FOR VISUALISATION OF AIR TRAFFIC DISRUPTIONS

WEBOVÁ APLIKACIA PRE VIZUALIZACIU PROBLÉMOV V LETOVEJ PREMÁVKE

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **MATEJ SLIVKA**
AUTOR PRÁCE

SUPERVISOR **Ing. JIŘÍ HYNEK, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2024

BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Bachelor's Thesis Assignment

154380

Institut:	Department of Information Systems (DIFS)
Student:	**Slivka Matej**
Programme:	Information Technology
Title:	**Web Application for Visualisation of Air Traffic Disruptions**
Category:	Web applications
Academic year:	2023/24

Assignment:

1. Study the issue of reporting flight disruptions. Study the existing processes and tools for processing data from these reports.
2. Study the general principles of data processing and visualization. Study the tools and libraries designed for this purpose.
3. Analyze the current process for handling and evaluating air traffic disruptions of the Kiwi company. Evaluate the shortcomings of the current process.
4. According to the results of the analysis, design a web application to process and visualize data from air traffic problems.
5. After consultation with the superviser, implement the proposed web application.
6. Test the usability of the solution in collaboration with Kiwi.

Literature:

- Few, S. (2006). *Information Dashboard Design*. O'Reilly Media, Incorporated.
- Johnson, J. (2010). *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Guidelines*. Morgan Kaufmann Publishers/Elsevier.
- Tufte, E. R. (2001). The Visual Display of Quantitative Information. Graphics Press.
- Internal documentations of the Kiwi company.

Requirements for the semestral defence:
Items 1 to 4.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor:	**Hynek Jiří, Ing., Ph.D.**
Head of Department:	Kolář Dušan, doc. Dr. Ing.
Beginning of work:	1.11.2023
Submission deadline:	9.5.2024
Approval date:	30.10.2023

Abstract

This thesis is focused on analysing the whole process of parsing flight disruptions for the Kiwi company. The main goal is to create a web application which will create statistical analysis on provided flight disruptions and visualise them. The application should ensure easier access to all disruptions from various databases. I created a frontend and backend, which will need to communicate together using API calls. While creating a web application, I simulated the behaviour of the Kiwi company services for development purposes. One of my most desired goals is to search in databases using the proposed application. The web application should serve aviation-providing company in managing flight disruptions and creating statistical analysis based on provided flight data. In this thesis, I documented the whole process of creating the service.

Abstrakt

V tejto práci analyzujem celý proces spracovanie letových problemov pre spoločnosť Kiwi. Hlavným cieľom je vytvoriť webovú aplikáciu, ktorá by štatistický spracovala poskytnuté data a vizualizovala ich. Aplikácia by mala uľahčiť prácu s rôznými databázami. Počas procesu vytvárania webovej aplikácie som sa rozhodol vytvoriť uživateľské rozhranie pomocou, ktorého by si uźivateľ dokázal jednoducho zadať filtry na vyhľadavnie z databáz. Vytvoril som frontend a backend, ktorý budú medzi sebou komunikovať pomocou API volaní. Jeden z najviac žiadaných cieľov práce je vyhľadavanie v databázach. Aplikácia by mala pomôcť firme, poskytujúcej letecké služby, s riadením letov a vytvoriť štatistickú analýzu nad poskytnutými leteckými datami. V práci som zdokumetnoval celý proces vytvárania služby.

Keywords

data visualisation, flights, flight disruptions, web application, air traffic

Klíčová slova

vizualizacia dat, lety, letový problém, webová aplikacie, letová doprava

Reference

SLIVKA, Matej. *Web Application for Visualisation of Air Traffic Disruptions.* Brno, 2024. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Jiří Hynek, Ph.D.

Rozšířený abstrakt

V tejto práci som sa zaoberal vytvorením webovej aplikácie. Prácu som vytváral v spolupráci so spoločnosťou Kiwi.com. Kiwi.com je spoločnosť predavajúca letenky a cestovné poistenie. Jej hlavným produktom je webová stránka, na ktorej si zákaznici môžu porovnať ceny od mnohých leteckých spoločností, ktoré predávajú lety po celom svete. Spoločnosť sa skladá z mnohých oddelení a ja som, v spolupráci s oddelením Disruptions Emails Automatisation, vytvoril túto bakalársku prácu.

Oddelenie sa zaoberá spracovaním leteckých problémov, ktoré môžu vzniknúť na letiskách. Najmä sa jedná o oneskorenie letu, zrušenie letu a zmena leteckej spoločnosti vykonavajúcej let. Tieto problémy sa z praxe nedajú odstrániť kvôli zložitému procesu vytvárnia letov. Celému tímu by v rámci práce výrazne pomohlo, ak by si mohli rýchlo a efektívne dohľadávať dáta z databáz. Spomínaný tím si často dohľadáva data z 5 rôznych databáz, pričom každá databáza má iné tabuľky, pristupové práva, schému zapojenia. Taktiež by mnohým členom tímu pomohlo, ak by sa nad danými dátami robila štatistická analýza, ktorá by pomahala pri tvorení manažerských rozhodnutí, a ktorá by poukázala na súvislosti alebo dlhodobé správanie dát v databázach. Dané problémy som sa snažil vyriešiť. Vytvoril som web, ktorý bol zameraný na vizualizáciu dát samotných, a to v jednoduchej a zrozumiteľnej podobe. Zo zobrazených dát som taktiež vytvoril grafy, ktoré automaticky analyzovali určité segmenty dát.

Celé riešenie problému sa skladá z frontendu a backendu webovej aplikácie. V rámci spoločnosti Kiwi sú už zadefinované databázy a backend, ktorý s databázami komunikuje. No kvôli vývoju webovej aplikácie a z právnych dôvodov som nemohol zverejniť spominané služby v mojej bakalárskej práci. Namiesto toho som dané služby naimplementoval a simuloval.

V baklárskej praći som zachoval architektúru, v ktorej bude produkt nasadený. Celý proces spracovania dát je teda následujúci. Vytvorí sa dotaz na aplikačnom frontende. Ten sa dopýta aplikačného backendu na samotné data a štatistický spracované data. Samotné data si dožiada z databázy a to pomocou databázového backendu. Po vyhotovení štatistickej analýzy sú data zaslané na aplikáčný frontend na vizualizáciu.

V rámci databázy som použil deväť mesačny záznam z databáz spoločnosti Kiwi.com. Tabuľky majú zachovanú rovnakú štruktúru, no data sú anonymizované a hešované. Implemntácia databázy bola realizovaná pomocu technológie PostgreSQL. Na prístup do databáz cez backend som použil jazyk Python a knižnicu SQLAlchemy. Prístup je zaistený pomocou vygenerovaných SQL dotazov.

Aplikačný backend bol implementovaný v jazyku Python, pričom štatistická analýza nad datami bola vytvorená pomocou knižnice Pandas. V rámci aplikačného backendu bola použitá technologia cacheovania a taktiež technológia vytvárania a spracovania requestov pomocou knižnice FastAPI.

Aplikačný frontend bol naimplementovaný pomocou jazyka React. Vizualná časť frontendu sa skladá z formulára, ktorý určuje dopytované data. Ďalej je na frontende zobrazená tabuľka, ktorá obsahuje niektoré kľúčové atributy. V rámci tabuľky je možne si rozliknúť detailný pohľad na dané data, ten zobrazuje všetky atributy aké sú v databáze. Taktiež sa na webovom frontende nachádza 5 grafov. 3 z nich vizualizuju agregáciu podľa zvoleného atributu. Konkretne sú to agregácie podľa databázy, z ktorej bol nahlasený letový problém, o aký typ letového problému sa jedná a agregácia podľa spoločností, ktorá vykonáva daný let. Ďalší graf kategorizuje dáta podľa času pred odletom. Posledný graf zobrazuje úspešnosť spracovania dát v čase. Dátová tabuľka bola vytvorená pomocou knižnice Shadcn, ako aj fromulár a detailný pohľad na dáta. Pomocou knižnice Nivo boli vytvorené grafy .

Spoločnosti Kiwi.com som podal už hotový produkt. V rámci testovania sme hodnotili ako daná aplikácia funguje. Aplikácia splnila všetky požiadavky spoločnosti. Dáta sú správne vizualizované a s fungovaním aplikácie sú spokojný. Tím, ktorému som aplikaciu podával, mal ale výhrady k dekoraciam na stránke a celkovo k spôsobu ako daná stránka vyzerá. Po úpravach frontendu webovej aplikácie spoločnosť Kiwi.com prijala moje riešenie problému. Webová aplikácia dokáže v rozumnom čase spracovať data z databáz a vizualizovať ich.

Web Application for Visualisation of Air Traffic Disruptions

Declaration

I hereby declare that this Bachelor's thesis was prepared as an original work by the author under the supervision of Mr. Ing. Jiří Hynek, Ph.D. The supplementary information was provided by Mr. Mgr.Radovan Lapár who helped me with technical part of bachelor thesis. I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Matej Slivka
May 6, 2024

Acknowledgements

My main gratitude belongs to two people. My supervisor Ing. Jiří Hynek, Ph.D. for deep patience with my ever-repeating questions and to my boss Mgr. Radovan Lapár who repeatedly explained to me many technological problems at our work.

Thank you for leading me when I got lost.

Contents

Chapter 1

Introduction

During the day, thousands of flights are taken across the globe. Aviation companies operate these flights with caution and aim to gain profit. It is a complex process, composed of getting many approvals from security controls, airports and business managers. During complex processes like this, aircraft might encounter troubles along the way. This may include any trouble varying from human factors to technical factors or business factors. This creates unwanted behaviour both for customers and for aviation companies. Kiwi.com is another company which needs to resolve these flight disruptions. It is an aviation service-providing company with the aim of automating the whole process of resolving issues related to flight disruptions.

As the whole process is complex, flight disruption reporting comes from many sources. They need to be automatically processed. Because many tools extract information from many sources, the data are stored in many databases. This may challenge uniting information extracted from these databases regarding one or more flight disruptions. Also, in the process of analysing data for future business plans, it is difficult to statistically analyse data from different databases with different schemes and tables.

The goal of this work is to easily access data from these sources without the need to access multiple different databases, visualise the data, and analyse them. In this thesis, I propose a web application that makes the whole process of extracting data from databases easier. Based on requirements, I wanted to make a simple and easy-to-use application that would automatically visualise data from different databases and create statistical analysis through many graphs. Users should easily filter out data using a form with predefined filters. This should ensure smoother operation when looking for data and their analysis.

In Chapter 2, I analyse flight disruptions. I dig into the whole process of creating flights, which sometimes results in flight disruption. I try to understand the whole process of scheduling flights, how different flights affect each other, who decides about flights, their paths and security. I mention different causes and sources of flight disruptions.

In Chapter 3, I take a look into visualisation of data. More precisely, the visualisation of flight disruptions. I understand the correct way of creating graphs and charts. I analyse how to parse information from different flight disruption sources. Lastly, I take a look into libraries for working with data visualisation and how they may help me in the process of solving this issue.

In Chapter 4, I analysed the whole process of resolving disruptions in Kiwi company. I take a look at how flight disruptions are categorised, how they affect passengers on their route, what is processing, and what part is it composed of. I analyse where the data is stored and what data needs to be mentioned to report flight disruptions. I pinpoint the weakness of the current process.

In Chapter 5, I propose a solution to the weakness of the current process. I came up with the design for a web application which would try to solve the problem of accessing data. I describe the technical part of the work. I propose frontend and backend and how they should communicate with each other. I describe the data flow. I design a mock-up with different parts, giving an understanding of extracted data about flight disruptions.

In Chapter 6, I describe the way I implemented my proposed solution from Chapter 5. I describe the technologies that I used. I depict the overall architecture of the project and the way I implemented separate project services.

In Chapter 7, I test my application. Testing was performed in multiple testing stages. I describe the way I created unit tests for my application. I explain the end-to-end testing performed on all services. Lastly, I describe testing with the users in the Kiwi company and how I implemented the changes they required.

Chapter 2

Flight Disruptions

"Disruption can be defined as an act of delaying or interrupting the continuity (Hyper Dictionary, 2003). However, in operational terminology, Clausen, Hansen, Larsen, and Larsen (2001) defines disruptions as a situation during the operation's execution in which the deviation from plan is sufficiently large that the plan has to be changed substantially. A disruption in flight operations takes place when the observed situation deviates from the planned situation and the deviation on operation is substantial. Disruptions may have minimal effect in some cases on the airline. In other cases, it can become severe, causing the airlines to delay, cancel or divert substantial number of flights and imposing substantial cost to them." [1]

While operating air traffic, most flights encounter no difficulties overall. Flights are usually departing on time and arriving on time; passengers have their correct seats, and there are no technical difficulties. However, sometimes air traffic providers and controllers might encounter some problems. These problems may cause some unwanted behaviour or troubles while operating air traffic. Such problems can not be avoided. Only one issue can create a chain reaction, which may lead to multiple other disruptions as they share the same airports of arrival or departure.

According to the Airline Disruption Management: A Review of Models and Solution Methods [23] *"The aviation industry has become a major player in the global economy. As people become increasingly dependent on air travel, the number of scheduled flights worldwide grows every year. The civil aviation passenger demand increased by 4.2% while capacity increased by 3.4% worldwide in 2019. Hence, demand appears to grow faster than capacity, suggesting great development potential for the aviation market. With the development of the aviation industry, airline planning and scheduling problems have attracted much attention, and most airlines benefit from advanced optimization methods. Sophisticated models and effective solutions have been developed for each stage of planning."* This only pushes further demand for handling any disruptions while operating airlines.

2.1 Scheduling Flights

The whole process of scheduling flights is complex. Flight, aircraft, and crew are scheduled months in advance. The whole process is described as follows: [8, 16]

1. Based on marketing decisions, the aviation company has to decide at what time they should schedule the departure and arrival of a given flight.

2. The fleet type that will execute the flight is chosen considering the demand for flight, aircraft capacity and availability.
3. The aircraft is chosen considering maintenance constraints.
4. After that comes crew scheduling, which has 2 parts:
   (a) First comes the crew pairing phase, which creates crew itineraries based on general regulations like maximum allowed working time or flying time per duty.
   (b) Second, comes crew assignment, assigning individual crew members to itineraries with the aim of minimising cost.

However, crew, aircraft, and passengers are loosely connected before the scheduling day. Actual crew members are assigned according to the crew's availability on the given day. Crew members, along with aircraft, may execute multiple flights during one day, which shows further dependencies of future flights on previous flights. Further, in the process of managing flights, the most important aspects are crew, passengers and aircraft. The process is also composed of ground staff (check-in staff, gate staff, ramp staff, and luggage staff), catering, and gates, but these resources are generally more flexible and less expensive than crew and aircraft and will only be considered briefly [16].

2.2 Flight Decision Makers

Operation control is performed by many elements. It needs to be ensured that smooth and safe flight operations will be performed. Unfortunately, not everything goes as it is planned. Flight depends on these decision-makers [16] who can change it significantly or even cancel it:

- Flight dispatch and following: dispatcher plays an important role in North America. He ensures flight safety and checks on the preparation and progress of a number of flights. The dispatcher communicates with other areas where problems appear. In Europe, the role of aircraft control usually only follows flights. Flight planning and dispatch are often planned outside of the operational area.
- Aircraft control: Manages central coordination, operational control, and management of aircraft resources. In Europe, it is divided into long and short haul. In North America, it is divided into geographical regions like North West, South West, West, South, South East, East, North East, and North.
- Crew tracking: This role is the management of crew staffing. Crew check-ins must be monitored, and crew pairings must be changed in case of delays or cancellations. A reserve crew must be issued in case of trouble. Usually, the crew is divided into cockpit and cabin crew.
- Aircraft engineering: This role is responsible for servicing and maintaining aircraft (as well as maintenance scheduling). Not all stations can do all types of maintenance, so changes to aircraft rotation can cause disruptions.
- Customer service: This role is not part of the organisational structure, but they are responsible for taking care of customers who encounter some inconvenience. Passengers need to be informed about disruptions, and in some cases, passengers need to be rebooked or ensured some meals or accommodation.

- Air traffic control: Typically, a public authority that creates rules for every aviation company. In Europe, it is the EuroControl. In North America, it is the Federal Aviation Administration.
- Duty manager: This role ensures overall coordination, ensuring that every group acts as one team. Most of the airlines leave the execution of such tasks to the Duty manager.

2.3 Reasons of Disruptions

As the process of scheduling flights is made in advance and is composed of many elements, there are a lot of possibilities for errors that may cause flight disruption. Outside of the pre-scheduled process, there are other factors that may affect flights. These may vary depending on weather factors, technical factors, and human factors (both on the crew side and also on the passenger side). Based on my analysis in Kiwi.com company, these might include the following:

- Ongoing strikes which may disable the whole airport for several days or maybe even for weeks. This may be caused by people demanding better salaries and calling attention to staff reductions and education overhauls [4].
- Problems with aircraft technology (damaged engine, wing or any other part of aeroplane). This can be caused by wildlife living in the area of the airport [25]. As the area of the airport covers significant space, which is usually on the city's outskirts, it creates enormous space which needs to keep wildlife out of it. Up to 3306 square kilometres of grassland are estimated to be contained at 2915 airports in the USA. Further, the storm drainage and flat landscape properties attract wildlife [21]. In case of damage, parts of the plane need to be scanned using methods like Ultrasonic testing [14]. In case of damage, the repair or replacement of damaged parts is needed.
- Fuel shortage. As a product of crude oil, the jet fuel is limited natural supply. However, the aviation industry is heavily dependent on this supply. With the current rising demand for travel [20, 23], aviation companies will have a hard time replacing it.
- Absence of crew members. This may be due to their health condition or due to their personal reasons. Furthermore, this issue might be more frequent as, for example, in the USA, the mandatory pilot's retirement leave is at the age of 65, with requirements of minimum flight hours for starting pilots going up. Also, retired military pilots, who have been an ample supply of new pilots for the aviation industry, have decreased in recent years [15].
- Delays of departure of the aeroplane. This may seem like an insignificant issue; however, all delays may cause flight cancellations. Furthermore, each minute of delay causes a substantial loss of money, which may be up to 100 euros per minute for European airlines [17]. With growing air travel, this may cause significant costs for passengers as well as for aviation companies.

  In the article Increasing stability of crew and aircraft schedules [8], it is proposed that while creating schedules for aeroplanes, the manager should take into account that the plane is going to be late. This is due to the fact that, according to the author,

flight disruptions are inevitable. Some techniques for reducing delays are scheduling with buffer time, swapping crew or aircraft or using a reserve crew to recover from the delay.

- Postponing flight operation due to previous problems related to flight operation within one airport [22].
- Schedule change of route of the aeroplane. This may be due to commercial reasons or due to a change of plans after the process of scheduling, as mentioned in Section 2.1 Scheduling Flights.
- Change of aircraft—this may be caused by aircraft availability or by technical damage caused to the aircraft.
- Flight cancellation. They may be caused by two types of reasons:
  - Cancellation beyond the control of the operator. This may happen because of severe weather conditions, equipment failure, or maintenance repair, which is necessary before the aircraft is operational.
  - Strategic flight cancellation. This happens when an airline operator cancels for economic reasons like low passenger bookings or when the airline operator cancels the flight to avoid further delays on other flights.

  [22]
- Whether forecast difficulties like rainfall, snowfall, frost or thunderstorms cause flight disruptions [3].
- Diverting of the plane to another airport, which may happen due to sudden medical emergencies on board [7] or to relieve a busy airport of traffic and divert it to a nearby airport. Another method is to dodge enemy attacks by diverting commercial planes to nearby airports due to safety reasons [2].
- Problems related to global pandemics (COVID-19). During the COVID-19 pandemic, governments enforced travel restrictions. This hugely affected air travel around the world. Depending on the country and its restrictions, air travel varies [24].
- Issues related to terrorist attacks. Hamas attacks on Israel or World Trade Center terrorist attacks. Most flights were changed, rerouted or cancelled due to safety reasons.

Further, flight disruptions are caused mainly by another plane which was disrupted, national aviation system delays, and extreme weather. However, there can be found a correlation between flight disruptions and the days of the week (Thursdays, Fridays and Sundays are less likely to have more disruptions), departure time and number of daily flights on a route. Also, the age of the aircraft that executes a given flight significantly impacts whether the flight is disrupted [22].

As these problems are unwanted by both passengers and aviation companies, diverting flights as a result of such problems creates an extra amount of fossil fuels, which harms the environment surrounding us [17]. All these troubles need to be resolved while providing the service of scheduling flights across the globe. These problems are called *Flight disruptions*. Flight disruption is any kind of act which may affect the original travel plan in a way that changes the scheduled plan.

2.4 Sources of Flight Disruptions

Flight disruptions can dramatically affect the passenger's route and create difficulties when transferring to other flights or create any difficulties in general. To report given disruptions, air traffic providers should notify or provide information about any flight disruptions or difficulties encountered during or before flight departures. Ideally, this information should be provided as soon as possible so that the passengers can react appropriately to a given change in time. Information regarding these flight disruptions can be reported from many sources like:

- Flight information from emails which is sent to the customer's email address regarding any kind of trouble which was encountered during the operation of a given flight. These emails should provide as much information about affected flights as possible, sometimes suggesting possibilities for passengers to manage his/her flight.

  Aviation companies can suggest or transfer passengers on alternative flights. They may also give a refund for flights. However, emails may not contain information about flight disruption, instead notifying passengers about existing issues. They might have a link to a website that contains information about flight disruptions, or they may attach a file that contains all necessary information about the flight.

- Flight information from companies—flight statuses and booking statuses can usually be checked on the website of a given flight provider. Passengers usually need to authenticate themselves to get information regarding their booking. This might be done by providing their PNR (Passenger Name Record) or/and passenger surname.

  Flight status should be accessible on aviation websites. Any further authentication should not be needed. However, this information is accessible only 3 days prior to the flight.

- Flight information from a third party—flight information provided by air traffic mapping websites. Websites like flightstats.com or flightradar24.com. These websites create virtual maps with updated information regarding each flight, as shown in Figure 2.1. They provide up-to-date information about ongoing flights mapped all around the globe using a GPS tracking system. The information obtained from these websites can be:

  - Airport Disruptions—chart of airports with the highest Delay index, with Delay index being the probability of Flight Delay. These indexes can go from 1 to 10—indicating a delay from 4 minutes to 175.

    The Lumo [12] delay indexes are a score from 1 to 10 given to each flight, indicating how "risky" a flight is with respect to being delayed. The score is intended to capture both delay frequency (probability of a delay occurring) and delay severity (how long will the delay be if it does happen).

    Quite simply, a delay index of 1 indicates near-certainty of less than a 30-minute delay, while an index of 10 indicates a near-certain delay of 2 hours or more; the numbers in between are essentially a weighted average of the likelihood of a delay of different magnitudes. Lumo calculates this index as a weighted average of 4 numbers:

    - The likelihood of a delay of less than 30 minutes (p0)
    - The likelihood of a 30-60 minute delay (p1)
    - The likelihood of a 1-2 hr delay (p2)
    - The likelihood of a 2+ hour delay(p3)
  - Flight information—this information includes carrier, carrier number, airport of destination and airport of origin. Scheduled arrival time and departure time also estimated arrival time and departure time.
  - Aircraft information—technical information regarding aircraft in use, such as aircraft type, registration and place of registration, serial number and time of use. Also, information regarding the terminal and gate which the aircraft departed from, its average flight time and great circle distance.
  - Operating information of aircraft—the scheduled flights of aircraft, altitude of aircraft, latitude, longitude, also ground speed, true airspeed, indicated airspeed.

- Flight information from airports—flight information provided by airports that track flights starting and arriving flights at the given airport. Usually, airports provide information regarding carrier, carrier number, airport of destination, scheduled time of departure and estimated time of departure. These websites also provide information regarding schedule changes and flight cancellations.

All these sources provide valuable information about flights, which can be used in the process of notifying and scheduling flights or transfers.

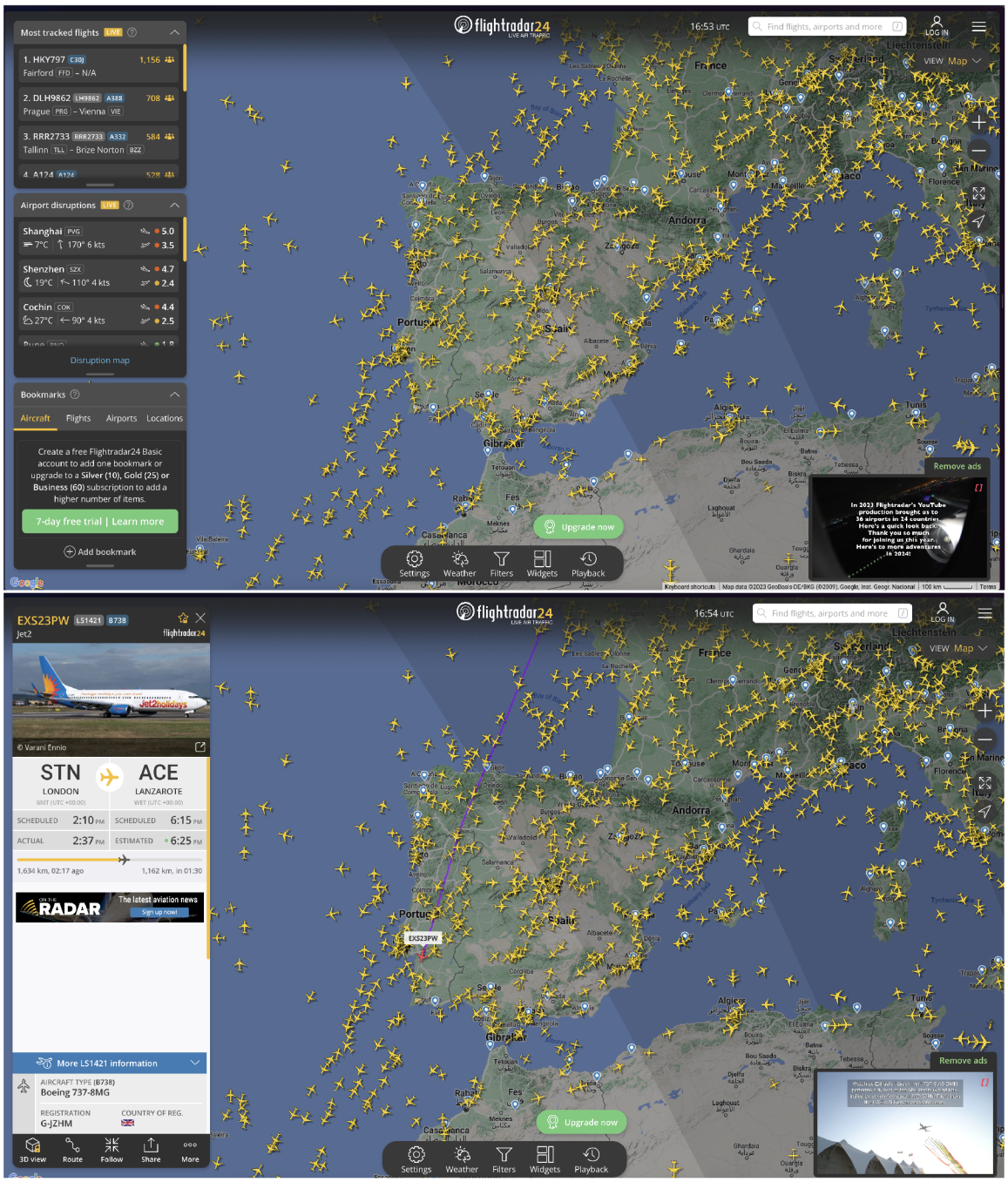

Figure 2.1: **Screenshots of flightradar24.com.** Both screenshots show a virtual map of flightradar24.com. The first screenshot depicts overall statistics. The second screenshot depicts the route of a single plane along with its statistics.

Chapter 3

Data Processing and Visualisation of Data

There are many ways of processing information from our surroundings and understanding the concepts. For example, people can feel pressure, which may explain gravitational force to us, we can hear effects like the Doppler effect, we can taste the pH of matter around us and many more. One of the most essential ways of accessing information is with our sight [9]. We can grasp many concepts with our eyes. For example, depth, colours, movement, etc. However, with our sight, we can grasp even more complex thoughts explained by human beings—we can read. Not only can we read text that can interpret human thoughts, but we can also read graphs and data that can interpret statistics and give us a higher level of information. Graphs can create relationships between data and create connections among them in many ways. Also, they can highlight some trends in data overall, as shown in Figure 3.1.

Data visualisation is a different approach to telling an idea. One of the reasons to use data visualisation is to process the amount of data that is being told to us. These days, we encounter thousands of pieces of information daily that tell different stories. To make sense out of all this information, data visualisation can comprehensively interpret this information. With correct visualisation, the reader can process huge chunks of data rapidly [11, 26]. It should push the viewer to think about the substance that the author wants to tell rather than to think about the methodology, graph, diagram, technology of graphic production, or something unrelated [27, 18].

3.1 Principles of Data Visualisation

Visualisation of data is an essential process for understanding the topic being discussed. Depending on the provided data, the visualisation can either set back or push forward the understanding of the given concept. The graphs should talk to the reader clearly and immediately [11]. However, it depends on the author of the work to choose the correct graph of display [18, 11]. With many graphs which have their pros and cons, there are few principles which can create understandable graph [18]:

- Use the right geometry—most geometries fall into categories: amounts (or comparisons), compositions (or proportions), distributions, or relationships. These categories are mostly displayed as follows:

Year	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	Annual
1990	127.4	128.0	128.7	128.9	129.2	129.9	130.4	131.6	132.7	133.5	133.8	133.8	130.7
1991	134.6	134.8	135.0	135.2	135.6	136.0	136.2	136.6	137.2	137.4	137.8	137.9	136.2
1992	138.1	138.6	139.3	139.5	139.7	140.2	140.5	140.9	141.3	141.8	142.0	141.9	140.3
1993	142.6	143.1	143.6	144.0	144.2	144.4	144.4	144.8	145.1	145.7	145.8	145.8	144.5
1994	146.2	146.7	147.2	147.4	147.5	148.0	148.4	149.0	149.4	149.5	149.7	149.7	148.2
1995	150.3	150.9	151.4	151.9	152.2	152.5	152.5	152.9	153.2	153.7	153.6	153.5	152.4
1996	154.4	154.9	155.7	156.3	156.6	156.7	157.0	157.3	157.8	158.3	158.6	158.6	156.9
1997	159.1	159.6	160.0	160.2	160.1	160.3	160.5	160.8	161.2	161.6	161.5	161.3	160.5
1998	161.6	161.9	162.2	162.5	162.8	163.0	163.2	163.4	163.6	164.0	164.0	163.9	163.0
1999	164.3	164.5	165.0	166.2	166.2	166.2	166.7	167.1	167.9	168.2	168.3	168.3	166.6
2000	168.8	169.8	171.2	171.3	171.5	172.4	172.8	172.8	173.7	174.0	174.1	174.0	172.2
2001	175.1	175.8	176.2	176.9	177.7	178.0	177.5	177.5	178.3	177.7	177.4	176.7	177.1
2002	177.1	177.8	178.8	179.8	179.8	179.9	180.1	180.7	181.0	181.3	181.3	180.9	179.9

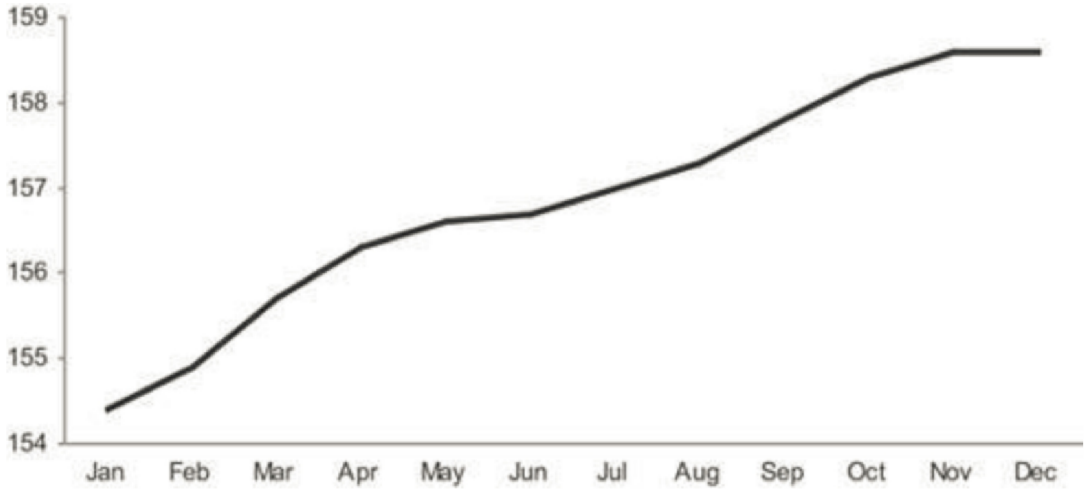

Figure 3.1: **Graph and table of data.** Both the graph and table show information regarding data interpretation. The graph might also pinpoint some data tendencies, like growth over time. This picture was taken from the book of Stephen Few [11].

- Amounts or comparisons are often displayed with bar plots—graphs A and B Figure 3.2, or with dotplots, even heat maps—graphs F Figure 3.2. However, bar plots often depict low data density. They may also be misleading as they always start at zero.
- Compositions or proportions may take a wide range of geometries. Traditionally, a pie chart is one option.
- Distribution—the most common geometry for distributional information is the box plot, which shows five types of information in one object—graph E Figure 3.2. The histogram is another robust geometry that can reveal information about data—graph B Figure 3.2. Violin plots and density plots are other standard distributional geometries—graph G Figure 3.2.
- Relationships—the basic scatterplot remains very effective, and layering information by modifying point symbols, size, and colour is an excellent way to highlight additional messages without taking away from the scatterplot—graphs C and D Figure 3.2.

It is recommended to always show data as they give a kind of proof of the whole concept of the graph that is depicted using them.

- Colour always means something—another easy way to add an extra piece of information to the graph. Colours can be used in 3 ways.
  - Sequential typically goes from dark to light, showing the scale of values distributed—graphs B and F Figure 3.2.
  - Diverging typically uses two different colours to show the opposites (black and white), with the middle part as grey, which shows a neutral element—graph E Figure 3.2.
  - Qualitative typically shows differences between two objects and does not depict any difference in values—graphs A and G Figure 3.2.
- Include uncertainty—not including uncertain and abnormal data or information can be a misleading element as it deforms a graph or diagram.
- Use description—if it is not apparent what the graph is depicting, use description. A few words can explain the graph and message that it sends. Alternatively, using infographics can help reading graphs.

3.2 Tools for Parsing Flight Disruptions from Sources

In this section, I want to dive into acquiring data from different sources because a part of this work is analyzing flight disruptions. As mentioned in previous Chapter 2.4, the data analysts or users can acquire data from at least 4 different sources. We choose different ways of parsing the data into complete flight disruption depending on sources. In the following subsections, I describe the mentioned ways of parsing.

3.2.1 Emails Parsing

Email parsing can be achieved by searching for a given HTML structure. Depending on the implementation language, we can choose the way we want to access data provided in email.

- HTML DOM tree structure—we can use HTML structures such as BeautifulSoup[1] or HTML5, which create DOM tree structure out of provided HTML code. By searching these structures, we can look for specific tag or IDs which identifies given data within the HTML structure. This can be accessed through the iterator.
- Regular Expressions parsing—we can create a regular expression which specifies the order of text and HTML structures in a given email. This way, we can create a template which always extracts data from a given position.
- Parsing from specific CSS selectors—CSS selectors can specify or highlight keywords which we want to parse from text. CSS selectors can help us by specifying searched data from HTML.

[1] https://www.crummy.com/software/BeautifulSoup/bs4/doc/

- Natural Language Processing—we can use this artificial intelligence, which will lead us to automatically parsing of a given HTML structure and outputting given attributes of flight disruption. This can be achieved by splitting the whole text into tokens and then composing the meaning of a given email. By understanding this composition, artificial intelligence can lead us to outputs which we want to get from email [19].

3.2.2 Accessing Data from Website

The other three ways of accessing flight disruptions are similar. They all use information from the website. It can be done in several ways:

- Connecting to prepaid APIs—some websites are willing to provide information through prepaid APIs. Websites may even create protected APIs. This is ensured by providing the client with an API key which authenticates the user within the system.
- Accessing website—another way of accessing data is directly from the website. Going to the website and searching for clues that could give us some information regarding flights.

3.3 Tools for Visualising Data

While creating the website, we are typically interested in 3 features of the website—the website's content, the style of the website and the functionality of the website. These 3 features typically correspond to 3 files that define a website:

- HTML file—which stands for Hyper Text Markup Language file. It corresponds to the content of the website [10].
- CSS file—which stands for Cascading Style Sheet. It corresponds to the style of the website.
- Scripts defining functionalities of websites. Typically Javascript [28].

These three files can define the whole website. However, for simpler website creation, there are frameworks that can be used to make web applications. Depending on the functionality that we want to ensure, we can choose from these popular frameworks:

- React[2]—which is very fast and has the feature of creating the components. Components may allow reusing of code. Also, React has a unique feature where it solves issues with re-rendering the page. When there is a change in the code, React updates only a specific part of the HTML code to update the web page.
- Angular[3]—is maintained by Google.com. It provides many already built-in features like routing, forms management, and client-server communication. In Angular, the Model-View-Controller pattern is followed, which means that the system links and binds the Model and the View. Changes made to the interface have an immediate impact on the objects of the application structure and vice versa.

[2] https://react.dev/
[3] https://angular.io/

- jQuery[4]—makes it much easier to use JavaScript on your website. jQuery wraps complex Javascript code into methods that you can call. jQuery also simplifies a lot of the complicated things from JavaScript, like AJAX calls and DOM manipulation.
- Vue.js[5]—is a Javascript framework for building interfaces. Vue extends standard HTML, allowing users to describe HTML declaratively based on JavaScript code. Vue tracks JavaScript changes and updates the DOM upon detecting a difference.
- Bootstrap[6]— is a CSS framework. Developers use Bootstrap because it automatically detects the size of the user's screen and creates a CSS layer responsively.
- Tailwind CSS[7]—is a CSS framework that helps developers define the style of HTML elements directly in the code. By naming class specifically, it defines the visual style of a given element.

As for the visualisation of data on web applications, we can use HTML elements, such as Canvas or SVG, to create graphs. Both create areas within the webpage where we can create elements for visualisation. The difference is the way they visualise it. Canvas creates raster pictures, while SVG is composed of vector-defined elements. Further, SVG provides functions for responsive handling of SVG elements, such as On-Click. Another way of showing data is by using a predefined API that returns our desired graph. For example, Google Maps[8] create an API which developers can connect in order to render maps on their sites. However, creating graphs ourselves might not always be easy, especially with more complex graphs. Therefore, there are several libraries for visualising data.

3.3.1 Libraries for Data Visualisation

- D3 library—it is an open-source JavaScript library. D3 library is a low-level approach for building data-driven, dynamic graphs. Composing a chart in D3 means composing it out of a variety of primitives. D3 is not a single monolith but rather a suite of 30 discrete libraries. D3 groups these modules together for convenience rather than necessity, so your tools are within reach as you iterate on your design. With input, such as CSV or JSON, it creates data visualisation [6].
- Google Charts—the most common way to use Google Charts is with simple JavaScript embedded in your webpage. The main goal of Google Charts is to create an easy-to-use system that is accessible to many, regardless of their technical expertise. It is a community-driven library meant to integrate with Google services environments such as Google Sheets or Google Analytics. Loading libraries and providing input data creates graphs. Then, creating with the correct id displays the Google Chart [13].
- Chart.js is a free, open-source JavaScript library for data visualisation. It supports 8 types of visualisations: bar, line, area, pie (doughnut), bubble, radar, polar, and scatter. Chart.js renders in HTML5 canvas. The library's goal is to create flexible graphs with a wide variety of modifications. It creates elements which can be displayed on mobile devices. On top of that, Chart.js has interactivity, which contains tools like hover effects, tooltips or clickable legends [5].

[4] https://jquery.com/
[5] https://vuejs.org/
[6] https://getbootstrap.com/
[7] https://tailwindcss.com/
[8] https://developers.google.com/maps

- Nivo is a free and open-source library built for using React. It is based on D3. It provides multiple sets of different pre-build graphs for visualisation. Nivo provides highly customisable elements. Pushing forward the developers' experience by providing modularity and reusability. Each component is encapsulated as a separate module. It is a developers-community-driven library for the visualisation of data.

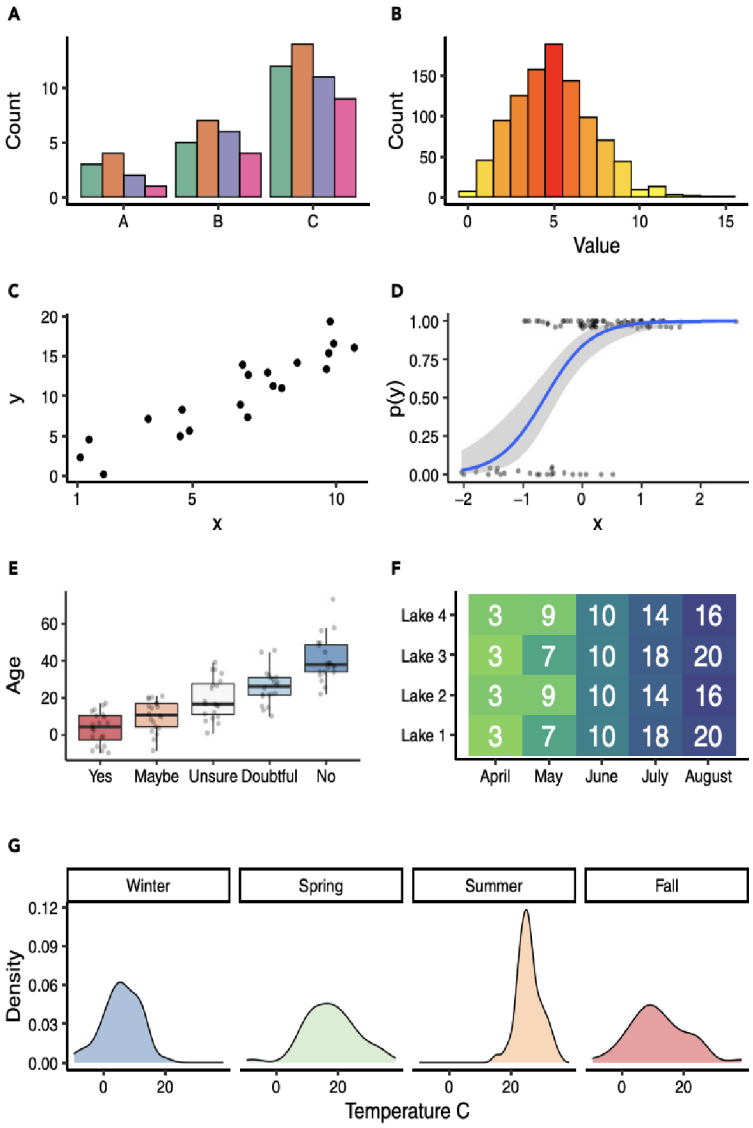

Figure 3.2: **Examples of different graphs.** In the figure, we can see 7 graphs depicting different ways of visualising data. *Graph A* is a bar graph showing differences between values. *Graph B* is a histogram depicting data distribution. *Graph C* is a scatterplot with dots representing the actual data. *Graph D* is a logistic regression with dots representing actual data. The line represents the model of regression. The grey area represents the confidence strip. *Graph E* represents a box plot which shows the age of respondents of a given question. The thick line in the middle represents the median. The vertical line represents the stretch of data with boxes representing the quartal, which divides the distribution of the given answer. *Graph F* represents a heatmap, which depicts visibility in lakes during different months of the year. Colour is adequate to the visibility of a given lake, creating a sequence. *Graph G* shows the Density plot of simulated temperatures by season. The data in these graphs are not real. These graphs were used as an example in the article Principles of Effective Data Visualisation by Stephen R. Midway [18].

Chapter 4

Analysis of the Problem

Kiwi[1] is an aviation service-providing company. The company's main product is a service (website) that allows its customers to search for and book flights from many different companies all around the globe. It serves as a tool which compares many different options on a market and offers solutions for passengers while managing their flights for them. To create an offer for customers, it searches the web to find the most suitable flights for passengers. These flights are then filtered, which can be done based on time, money, earliest departure or number of transfers. These days, with the increasing demand for airline services, a flight searching system is more needed than ever.

While managing the flights, Kiwi company might encounter many problems that may affect their customers' experience, primarily the problems related to air carriers and flights.

4.1 Categorizing of Flight Disruptions

Kiwi company categorizes all flight disruptions into five different categories depending on how they affect a given flight and how these flights can be parsed later in the process of addressing flight disruptions. These categories are:

- Delay—which includes delays of the estimated time of departure and estimated time of arrival of an aeroplane on its route.
- Schedule Change—which includes any kind of change in scheduled departure time or scheduled arrival time. These include earlier or later departures and arrivals of flights.
- Flight Replacement—which includes changing of aircraft. This might also change a flight number (a number which identifies a given flight within one air traffic-providing company). Changing one of the previously mentioned attributes, along with changing the scheduled departure or arrival time, is also classified as Flight Replacement.
- Flight Split—which includes splitting flights into more flights. The route of the flight split has to be somewhat similar to the route of the replaced flight. For example, flights going from London to Dubai can be split into two flights. London to Vienna and Vienna to Dubai. These may include splitting flights into even more than 2 flights.

[1] https://www.kiwi.com/

- Flight Cancellation—which includes any kind of flight cancellation caused by any kind of problem (airport strikes, business-related problems, technical difficulties, weather difficulties, safety difficulties, political difficulties, etc.).

Depending on the type of flight disruption, Kiwi company creates an adequate response to a given problem.

4.2 Processing of Flight Disruptions

Flight disruptions reported from different sources are usually parsed automatically. This includes acquiring disruption from the source, validating the data, reporting given flight disruption and updating databases, alternatively creating a notification for manual processing in case of failure. The whole process is also described in Scheme 4.1.

All parsed data have to go through data validation in the first place. Disruptions validating uses a list of plug-and-play prepared validators that can be enabled or disabled for all reported disruptions or just by disruptions from a specific source.

Here are the examples of some of the existing validations:

- Checking the reported flight disruption—this validation checks whether flight data reported in flight disruption are complete. For example, if there is no missing departure of the aeroplane. If there is one, this validation will not go through.
- Validate Carrier IATA—IATA Carrier code is a 2-letter code given to a particular aviation company which operates under this code. It is assigned to a company as a sort of identification[2]. This validation checks whether the reported Carrier IATA in flight disruption is a number. Valid Carrier IATA should contain a letter.
- Validate Carrier Number—carrier number identifies flight within one aviation company. This validation checks whether the reported Carrier Number is a 1 to 4-digit number.
- Arrival or Departure Date and time—this validation checks both departure and arrival times. They have to be reported using the correct data class structure.
- Validate PNR—this validation checks if PNR is not too short or if it is missing from reported flight disruption.
- Source Airport and Destination Airport—this validation checks both reported IATA codes of airports. They need to be 3 capital letters.

If data validation is successful, then the flight disruption is forwarded to automatic processing, which notifies the customer and takes action depending on the given disruption and flight protection that the customer has had required. The customer is always notified about flight disruptions which happen to his/her flights. In case of more significant changes which may affect customers' flights, the customer may be automatically re-booked.

In case the automatic process fails in some step of automatic validation, the flight disruption is reported to the Manual resolving queue, where they are supposed to be resolved by people manually with a given air-traffic provider.

[2] https://www.iata.org/en/publications/directories/code-search/

Flight disruptions can be reported data via different ways of data access. These sources are:

- Flight Information from emails—emails which are sent to the customer regarding any kind of trouble which was encountered during the operation of a given flight.
- Flight Information from a company—flight status and booking status can usually be checked on the website of a given flight provider.
- Flight Information from third party—flight information provided by air traffic mapping websites.
- Flight Information from airports—flight information provided by airports which track flights starting and arriving at the given airport.

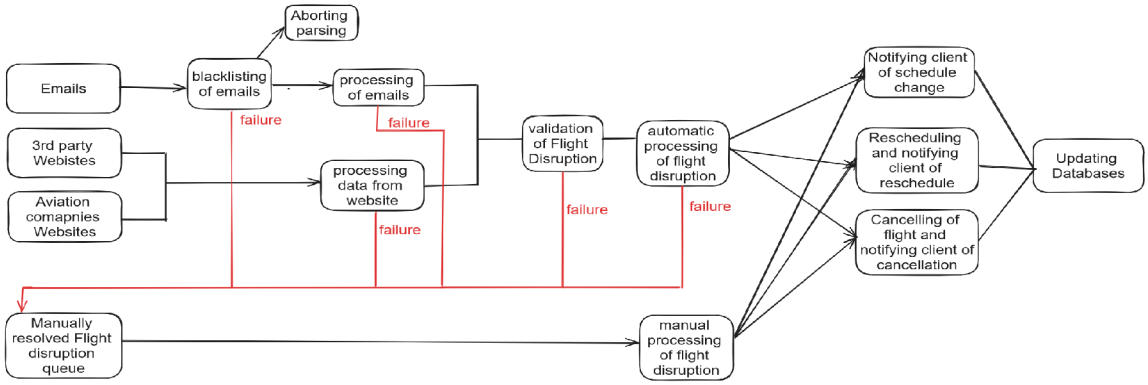

Figure 4.1: **Scheme of processing flight disruption** Scheme shows the process of handling Flight Disruptions from sources of flight disruptions up to the point of notifying clients and updating databases. Note that in case of failures, the process is being handled manually by people who need to resolve the issue. The desired path leads to automatic processing.

4.3 Internal Storage and Data Access

Information about flight disruptions is stored in multiple databases. Based on the data access mentioned above.

4.3.1 Emails Database

There is a pursuit to parse every email which is reported to the Kiwi company automatically. If this email has any kind of information regarding flight disruption, then it should be reported and automatically parsed. Kiwi has a code segment dedicated to automatically withdrawing critical information from emails. This information is:

- Departure Airport—the airport from which the given flight should depart, alternatively, the IATA code of the given airport.
- Arrival Airport—the airport to which the given flight should arrive, alternatively, the IATA code of the given airport.

- Arrival date—date when the given flight arrives at Arrival Airport.
- Departure date—date when given flight departs from Departure Airport.
- Arrival time—time when given flight arrives at Arrival Airport.
- Departure time—time when given flight departs from Departure Airport.
- Reservation Number—a six-characters long identifier, which identifies a given flight reservation in the flight provider's system.
- Carrier Code—code of aeroplane carrier which identifies a given flight provider company within air traffic. The Carrier Code and Carrier Number should identify a given plane within the entire air traffic area on a particular day.
- Carrier Number—one to four-digit number which identifies a given flight within one company. The Carrier Code and Carrier Number should identify a given plane within the entire air traffic area on a particular day.

The IATA code of an airport or carrier is a group of letters and numbers identifying a given airport or carrier within air traffic. Carrier IATA codes are 2 symbols containing at least one capital letter. It can contain numbers. Airport IATA codes are 3 capital letters. PNR is the Passenger Name Record. It identifies passengers' reservations within one system. There can be multiple people listed under one PNR.

All this information, which was reported this way, is stored in the Emails Database. The database also stores information regarding the time of flight disruption reporting and time of flight disruption parsing.

The database also stores information regarding email, which was parsed like sender email address, receiver email address, or subject of email. Some companies send flight disruption information in attachments, so we also need to save attachment data.

Lastly, the Emails Database stores data regarding reservations, which can identify given bookings within Kiwi company or within the air service providing company.

4.3.2 Company Provided Disruption Database

This database is for the purpose of marking flights with Kiwi company partners. Not all flight disruptions are accessed via Email notifications. Some flight disruptions are accessed through information given on flight provider APIs. To access this information, we communicate with our partners. We need to look at our providers' flight status APIs automatically. This information is similar to data extracted from the Emails Database.

As every flight disruption should be notified via Email, not every checking of Booking can create a notification for the internal system about the flight disruption (email already created notification). This way of accessing data is not sufficient. However, it may be good practice to have data like these as a secondary check for flight disruption about critical and essential flights to the company.

This database does not need any further information regarding flight disruption. The reported disruption itself is enough to identify the affected Flight. Other than disruption itself, the database stores information regarding reservation and booking itself.

4.3.3 Third Party Disruption Database

There is a pursuit to get notified about as many flight disruptions from as many sources as possible. Other than searching for data online on Air traffic-providing sites or from Email notifications, we can get information about flights from third parties who share information about flights. We can get information about flights from:

- Airport websites—airports usually provide information about current flight statuses and delays online.
- Air traffic radars—some sites are focused on mapping the current state of flight which are flying across the globe.
- Twitter—some companies notify their users about flight disruption on Twitter.

This database, similar to the Emails Database, extracts from given sources the arrival date and time, departure date and time, origin airport, destination airport, and carrier and carrier number.

4.3.4 Manually Resolved Database

The previous three databases store data they find from different sources. Usually, data that are parsed that way are later processed automatically to notify the passenger. However, data parsing is not always successful, and some data and flight disruptions need to be checked manually. These manual checks are performed by people. They have to go through emails or online bookings and make notifications or make changes to reservations. In order to avoid losing track of these changes, we store information about these changes in the Manually resolved database. Notification requiring flight disruption addressing can be created automatically if flight disruption does not go through validation in automatic processing.

4.3.5 Reservation Database

This database stores information about flights, which should be up to date. Flights mentioned in this database are compared to reported flight disruptions. This determines the action the system takes in order to address a given problem.

4.4 Shortcomings of the Current Process

In the Kiwi company, there is a considerable pursuit of automatically parsing every flight disruption that can be found online. Data are accessed through many sources. In case of any flight disruption which may affect flights significantly, Kiwi takes action to ensure the smooth operation of flights.

However, in some cases, automatic flight disruption handling results in failure. They have to be resolved manually by people working with their computers. In Figure 4.1, I present the whole process of parsing flight disruptions. These people might encounter trouble fetching actual flight data from the database and they might encounter trouble looking for flight disruptions themselves. As flight disruptions are stored in 4 different databases, they might have some trouble fetching data from databases as it requires some technical expertise on how to search in a database. An application which could ensure the

visualisation of data from these 3 databases, along with the database of Manual Disruption and data from the Reservation Database, could help them in processing of flight disruption for customers. Along with actual data themselves, statistical evaluation from these fetched data can help manage the purposes of the Kiwi company. These data might help in the decision-making of strategy of evaluating flight disruption automatically. Furthermore, even for people with technical expertise, there is no unified way of visualising data from these 4 databases, and it is time-consuming to search for data from these sources.

To sum up, developers, people who manually book flights, managers, and team leaders need an application to visualise data from databases. People who manually book flights would use it only to search for specific flights and check the flight status. Developers, managers, and team leaders would do the same, but they also need to use the application to check aggregated data. For example, most used carrier aggregation provides business value information. Also, data tendencies like low automatic parsing would be helpful for developers. Developers also need to check whether parsed flight disruptions were made before actual flight departure. Then, developers need to check the status of the flight disruption sources and flight disruption types. All these functions could help developers, managers, and team leaders investigate the problems connected to maintaining the proper working of their code.

Chapter 5

Design of Web Application

Based on my analysis, I decided to create a web application that can access 4 disruption databases and visualise data which were fetched. The application should be easy to use, with a simple interface for searching reservations in databases. This application should help users who resolve flight disruption manually by providing actual information regarding flight status. It will make searching for flight status faster as people who resolve these issues do not have to look for flight disruption from many different databases. Instead, they just need to use the application and fill out the provided form. The application should output every disruption from every source that it can get. Furthermore, the application should perform statistical analysis and data visualisation on fetched data. This process can serve the purpose of managing strategies for future purchases or for future code refactoring.

5.1 Data Flow of Web Application

In the following sentences, I will describe the ideal data flow and the intended functioning of the application. For illustration, I present Figure 5.1. First, the application user needs to select out (on the frontend) which kind of data he wants to fetch from all databases. This will be done by filling out the form inputs corresponding to filters (for example, one input for carrier, one input for carrier number, etc.). After submitting the desired request to the web application, the fronted will create an API call to the web application backend. It will have to check the inputted data and fetch data from databases. The application backend needs to fetch data from 4 different databases. These databases are not accessible directly, but the application backend needs to connect to them through their predefined backend. Each database needs to have a separate call to withdraw data from it. These data are then fetched back to the web application backend. In the application backend comes raw data parsing. It will create the JSON dictionary needed for raw data visualisation. Also, the web application backend needs to create statistical analysis on the provided data. For each graph, there needs to be a correct JSON, which will be displayed. Statistical analysis for each type of graph is as follows:

- Pie Graphs—as this graph is supposed to depict the composition, as mentioned in Section 3.1, we need to aggregate flights based on their wanted properties. For each type of graph, it is:
  - Automatic vs Manual—this aggregation defines the way of processing flight disruption. It can be either manual or automatic. Implementation-wise, this means going through all fetched data and counting these 2 outputs.

  - Types of disruption—one of five disruptions mentioned above in 4.1 plus one attribute none, which means the disruption was not created. Implementation-wise, this means going through all fetched data and counting how many properties belong to each category.

- Bar Graphs—one bar graph is supposed to depict the most frequent carriers which organised the flights mentioned in fetched data. The other bar graph depicts the time between the parsing of flight disruption and the actual flight time.
  - Implementing top carrier graph means going through all fetched data and counting each new carrier.
  - Implementing time between the parsing of flight disruption and actual flight time means dividing the timeline into positive and negative values. Then, divide the timeline into defined categories, which will represent a time of parsing prior to or past the actual flight departure (for example, 5 hours prior to the flight, -2 hours after the flight, etc.). Then, manually go through all fetched data, categorise them into these categories, and define the success rate of parsing.

- Line Chart—it is supposed to depict the success rate of automatic parsing over a time period. Implementing this means defining several time periods, which will be depicted on the graph and calculate the success rate for each of these periods (calculating how many flights were parsed out of all flights in that time period). Then, put all the columns together, which will create the graph itself.

During filtering, I need to take into account that the same disruption might be reported from different sources. As these data are redundant and can create misleading graphs, I need to filter them out. Then, after all these calculations, we create a JSON file with data. The data are sent to the frontend, where they are visualised in graphs. In Figure 5.2, I present a mock-up of the application.

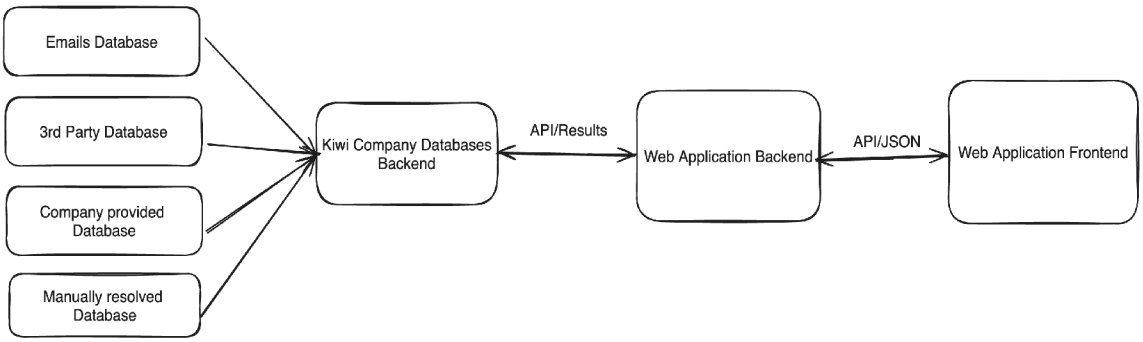

Figure 5.1: **Scheme of application data flow.** Scheme shows data flow among elements which will compose the web application. It is needed to communicate with the Kiwi company backend, which will fetch data from each database. Filtering and data visualisation are made later in the process.

Out of the four parts which are mentioned in Figure 5.1, there are already defined databases and the Kiwi company database backend, which I do not need to implement. My implementation is composed of the web application backend and frontend. The backend needs to communicate with the Kiwi company backend, which needs to create API calls for databases to withdraw the needed data. Then, the data need to be filtered and sent to the frontend, which needs to be designed to update information upon receiving them.

As for my backend API calls, I need to create the main API call, which will fetch all data that will be visualised on the frontend. This will be based on filter values provided from the frontend. However, these data need to be inputted based on convention. For example, inputting the date and time might be critical. I suggest inputting the date and time in the format YYYY-MM-DD-HH-MM. These data need to be checked on the backend and provide answers in case of wrongly submitted data. Other than that, I should create pagination for the data table, as sending all information to the frontend might overload the application. I need to send an adequate amount of data to the frontend. This means creating an API to get to the next and previous page. Other than that, I should implement a detailed view of a flight disruption as it may be better for visualisation than putting all data in the data table. The detailed view should be accessible directly from the table.

This system should be expandable easily. For example, creating a new graph should be done by simply writing a new HTML component on the frontend side. On the backend side, it means fetching one more set of statistically analysed data, which will be displayed. Another possible expansion is adding a new filter to the submit form at the top of the web page, see Figure 5.2. This filter will be another value that needs to be checked and another value that needs to be sent to the database.

5.2 Mock-up of Web Application

For the mock-up, I created the design in Figure 5.2. The web page should be composed of filters, which can be seen at the top of the page. The user should be able to filter out data based on different attributes of flight disruption (for example, carrier, source of flight, parser which executed the parsing, etc. See the mentioned Figure 5.2.). Upon entering data, the frontend will make a call to the backend, which will then fetch the given data, filter them and send them to the frontend as described in the previous Section 5.1. Then, the frontend will visualise fetched data along with multiple graphs created from fetched data. Mentioned graphs are:

- All data table—this all data fetched and visualised as it is needed to see the actual data for people working with manual processing to make decisions based upon these fetched data. Also, one of the main functions of this application is to visualise databases. This table should visualise all data fetched from the database. In case a huge amount of data is fetched, it needs to be paginated. As depicting all data might not be the best solution, the table should have a button which will pop up a detailed view of flight disruption.

- Automatic vs manual—this graph shows the composition of automatic versus manual processing of flight disruption. I chose the pie chart as it is according to Section 3.1 on visualising data—it is good practice to show the composition with a pie chart. This chart should show at what state automatic processing is and the overall performance of manual vs computer processes. Despite Stephen Few mentioning that people do not have the ability to compare angles to each other [11], I prefer research mentioned in Section 3.1. Pie graphs create an intuitive understanding of the composition of given data. This also goes for the next pie graph.

- Types of disruption—this graph should depict one of five flight disruption types, which are mentioned in Section 4.3.1 of emails database. Similarly to the previous graph, I chose the pie chart as it is good practice to show the composition with a pie chart.

This graph can help determine the impact of the world around us on the performance of our code. For example, in case of multiple flight cancellations from one airport, we can determine troubled airports which have a high possibility of cancelling flights in the future.

- Top carrier—this graph shows the aggregation of fetched carriers. It creates the bar plot, as in the previous Section 3.1 I described that it is good practice to show comparison with a bar graph. It shows multiple instances of a key measure [11]. This graph should determine which carrier is most used and has the most value for the company and the biggest business impact.

- Success rate—this graph depicts success rate over time. I used a line chart as it highlights long-term trends and creates comparison. A line chart, as the pattern formed by one or more lines in a line graph, can represent a great deal of information as a single chunk [11]. This chart should show the performance of disruptions through time. It serves as a review for the development team to see how they are doing compared to the past.

- Time between flights' departure and parsed time—this graph shows the difference between the time of actual flight and time of parsing on our side. If this graph goes to negative values, it will suggest that the processing time of Kiwi company is too late or in advance. I choose the bar graph as it is decent for comparison, as mentioned in Section 3.1, and shows multiple instances of a critical measure.

Figure 5.2: **Mock-up of web application.** Figure shows a mock-up of a web application with all its components. There is a filters section at the top, and beneath, there is a table of all fetched data. On the sides, we can see graphs that depict the statistical analysis of fetched data and its visualisation.

Chapter 6

Implementation of Web Application

The implementation chapter is divided into two sections. The first section describes which technologies were used in the process of writing this work and information regarding them. The second section describes the overall architecture of the system, algorithms, and how I handled data parsing and the overall connectivity of the software architecture components.

6.1 Used Technologies

- For creating a database, I used PostgreSQL[1] as it is also used in Kiwi company, and I wanted to simulate a real environment as much as possible.
- For both application backend and Kiwi-backend, I used Python. As a Kiwi company works in a changing environment, I wanted an agile programming language that can adapt quickly. It also provides many libraries for working with data, statistical analysis and databases. I used the latest stable version 3.11, as it is optimized, it does not contain bugs and most of the libraries are made for newer versions.
- For connecting to database, I used SQLAlchemy[2]. SQLAlchemy is the Python SQL toolkit. It contains functions that connect effectively to databases.
- For parsing connections on server sides, I used FastAPI[3]. It is a modern, fast web framework that creates API in Python languages 3.8 or newer. It is fast to code using Python decorators.
- For working with data, I used Pandas library[4]. Parsing large data, it creates an object DataFrame with many fast methods for aggregating, parsing, changing of rows and columns within the table.

[1] https://www.postgresql.org/
[2] https://www.sqlalchemy.org/
[3] https://fastapi.tiangolo.com/
[4] https://pandas.pydata.org/

- For caching data on application backend, I used Python CacheTools library[5]. It uses a simple decorator, which puts data into temporary memory and can help you store data for a temporary time. This way, it creates a faster response, ensuring faster project speed. CacheTools provides multiple caching techniques which determine how are data temporary stored. It is based on the logic of queues—First in First out, Least Frequently Used element, Least Recently Used element, Most Recently Used element, Random Replacement, Time to Live or Time Aware Least Recently Used element. This project uses the deletion of the least recently used element.

- For frontend implementation, I used React[6], which allowed me to create the user interface and use libraries for pre-defined components. I implemented frontend using Next.js framework[7]. It is only a dev-dependency in which I implemented the whole frontend, and I did not use server-side rendering.

- For more friendly styling, I used Tailwind[8], which helps you to define the styles of your HTML elements. Simply by defining the class names of elements that compose actual Cascade Style Sheet attributes.

- For easier deployment, compatibility and transportability of given app, I used Docker[9] and for managing the created Docker containers between each other I used Docker-compose[10].

- For versioning and source code managing, I used GitLab. The whole project can be accessed from link[11].

- For documenting and creating a thesis, I used Overleaf[12].

6.2 Architecture

The architecture used in this thesis follows the Scheme of application mentioned in Figure 5.1, about the Design of Web Application. The main goal of practical work was to create the application's frontend and backend, which would communicate together. In deployment, the application backend should communicate with the Kiwi backend for databases. Then, the Kiwi-backend should communicate with databases. However, due to legal reasons and due to the scope of this work, I am not provided with Kiwi-side services (Kiwi-backend and databases). Therefore, I simulate Kiwi-backend and databases themselves.

As for components of the architecture

- Application frontend is a web which enables users to search, view and filter data provided by the application backend.

[5] https://cachetools.readthedocs.io/en/latest/
[6] https://react.dev/
[7] https://nextjs.org/
[8] https://tailwindcss.com/
[9] https://docs.docker.com/engine/reference/commandline/cli/
[10] https://docs.docker.com/compose/
[11] https://gitlab.com/ApetorSkol/disruption-viewer
[12] https://www.overleaf.com/

- Application backend receives requests from application frontend. Upon receiving a request, it creates a request for the Kiwi-backend asking for data. Upon application, the backend receives data from the Kiwi backend. It parses the data and, gives them new attributes, and then creates statistical analysis for graphs on the frontend. All this information is sent back.
- Kiwi-backend listens for communication from the application backend. Upon receiving a request from the application backend, it creates a query, which is then sent to the database asking for data. Upon receiving data from the database, it sends fetched data back to the application backend.
- Database stores all data regarding flight disruptions and metadata regarding flight disruptions. Upon receiving the query from the Kiwi-backend, it responds with filtered-out data. In this work, I got data only from one database. However, in deployment, there will be 4 databases accessed by one Kiwi-backend.

6.2.1 Database

The database stores all the data regarding flight disruptions that have happened. For the development of this work, the Kiwi company provided me with comma-separated value sheets that represent given information regarding flight disruptions. In total, I received over 9 months of mock-up data from databases. Comma-separated values are stored in two files. One file contains information regarding all flight disruptions. This information is:

- ID of given disruption which identifies disruption within the internal system.
- Source of parsing of given flight disruption. In this case, it can be either emails or API sources.
- Type of flight disruption, which means whether the given flight disruption is a schedule change or flight replacement.
- Time of creation of Flight Disruption.
- Status of flight disruption, which decides whether flight disruption was processed, failed, or skipped in case of unwanted emails.
- Whether processing was automatic or manual.
- Reservation code which identifies given flight with flight provider company or website.
- Carrier, carrier number, time and place of both destination and origin airports. As flight replacement disruptions might change the flight route significantly, this includes information before and after a change (original segment and revised segment).
- Lastly, the mentioned file contains a foreign key which identifies a given flight disruption within other comma-separated values file provided by Kiwi company.

The second comma-separated values file contains metadata regarding emails, which triggered the whole process of parsing emails into flight disruption. They have the primary key, which is mentioned in one flight disruption from the previous file, and connect the flight disruption with the mentioned email metadata (every email has a flight disruption, but not every flight disruption has an email). Other than that, the file contains:

- Email address of the sender of the given email.
- Email address of the recipient of the given email.
- Subject of given email.

As in reality, the provided data are not stored in comma-separated value sheets, but rather in databases, I created two tables and filled them with provided data. As mentioned in Section Used technologies 6.1, the whole database is made using PostgreSQL. It stores all data and awaits queries using Structured Query Language (SQL), to which it responds with data.

6.2.2 Kiwi Company Backend for Databases

For the Kiwi Company Databases Backend, I created a server. It is a service which parses HTTP GET connections from the application backend. Using FastAPI, it filters out given flight disruptions based on provided requests. It gets a query that is checked using regular expressions. The query should contain each filter that exists exactly one time. However, there may not be any value assigned to it.

The regular expression also checks for the keyword `" or "` which is used in filtering. It extends one filter to multiple options. For example, a request can define that it requires a flight disruptions that involves one of several airports or carriers. While searching carriers with code `"XB"` and `"CD"`, instead of writing `"XB"` and `"CD"` to the search each time, we can write `"XB or CD"`, which fetches all data that have carrier either `"XB"` or `"CD"`.

This way, the regular expression returns all filters that I need in order to filter data from the database. The service creates a query for the database. Firstly, the backend selects all attributes except primary and foreign keys mentioned in the previous part 6.2.1. Then, it joins both tables that are provided on the key. Lastly, iterating through all provided filters, it defines values in given columns of data. In case of no filters are selected, it simply returns all data. Upon creating a query, it is sent to the database. The database responds with filtered-out data. Fetched data are parsed using the Python Pandas library and sent to the application backend. This is the whole part of simulating Kiwi side services. The database backend should just return all non-aggregated data regarding flight disruptions for further processing.

6.2.3 Application Backend

The application backend awaits requests from the application frontend. Upon receiving a request, it is processed, data are fetched, and a statistical analysis of given data is created, which is then sent to the frontend for visualisation. The application backend can process 2 different requests. First is the overall data processing, which includes statistical analysis using the Pandas library. The second request is for pagination purposes. It is not effective to send all fetched data to the frontend for visualisation as they may contain over hundreds of thousands of data lines for visualisation. This way, the frontend may freeze or shut down. Therefore, I implemented pagination, which sends a maximum of 50 lines of data to the frontend at a time.

Creating New Search

As for creating a new search, firstly, the application backend checks whether the query which was sent is valid and contains all required filters. It is performed using regular expression. Similarly, as in the previous Subsection 6.2.2, it takes into account the possibility that one filter might ask for multiple values using the keyword `" or "`. In case a query is not valid, the exception is thrown. If it is valid, then the same request is sent to the Kiwi backend, demanding the data. After the resulting data are fetched, they are stored in Pandas DataFrames for further processing. The extra attribute is attached to data for further processing (the mentioned attribute is parsing time compared to departure time, which determines whether parsing of flight disruption was on time or not. This attribute is not put into the database for optimising as the application backend and web application should be the same in Kiwi production as they are in this thesis, and changing the database would be inefficient). Then, filtering on the backend side is made based on the provided attribute.

The statistical analysis begins after we filter out all the data we need. Using Pandas methods, I aggregate and create DataFrame to determine the analysis of flight disruption types. Using the same method, I get similar outputs while aggregating the Source of flight disruptions and Revised Segment Carriers of flight disruptions. In the case of Revised Segment Carriers of flight disruptions, I can get over 30 results of carriers which, after visualisation, create a chart which is not good visually. Therefore, in case there are more than 30 results, the last n-30 results are collected into one category, `"others"`, and visualised this way. The mentioned graphs are displayed in Figure 6.1.

Another graph for visualisation is the success rate graph of processing flight disruptions automatically throughout the selected time. In this case, flight disruptions can be categorised as parsed, failed (due to error or non-existing code) or skipped (in case of unwanted email). As it is not possible to interpret continuous time on a finite screen, I decided to create a time window in which we calculate successfully parsed flight disruptions, skipped flight disruptions and failed flight disruptions overall flight disruptions. It is performed by selecting maximal and minimal parsing time out of all flight disruptions. This creates a period of time, which is then divided into 30 time frames. For each time frame, it is manually counted how many flight disruptions belong there. After processing, this creates a list of values that define the success rate graph. Figure 6.3 depicts the mentioned graph in the upper part.

The last graph that I implemented is composed of an added attribute that defines how much Kiwi parses flight disruptions in advance. It is simply a list of predefined values and counts of attributes belonging to them. For example, the list contains value 8h. 8h is a category of flights that happened within a time window of 8 hours prior to the flight departure. On the graph, 8h has a number assigned to it, which defines how many flights happened a maximum of 8 hours prior to flight departure. Figure 6.3 depicts the mentioned graph in the bottom part. After all these statistical analyses are made, the results, along with the first 50 rows of data, are sent to the application frontend for visualisation. After statistical analysis is performed, the raw data, along with graph data, are put into one dictionary. This dictionary is then sent to the frontend for visualisation.

Pagination

As for requests responsible for pagination, it follows a similar process of sending data to the application frontend. As mentioned before, it checks for correct queries using a regular expression, fetches data from the Kiwi backend, adds additional value to the Dataframe and sorts by it. However, in this case, it only responds with 50 rows of data. Depending on the page number, the 50 rows which are sent back will vary. The visualisation of the data table can be seen in Figure 6.2.

I implemented caching as creating statistical analysis, fetching data from the backend, and adding extra attributes to Pandas Dataframe may take a lot of time. Each time a request is made, CacheTools automatically saves data to temporary memory, which is used as a response. This way, in the case of similar or identical calls, cached data will make data fetching faster.

6.2.4 Application Frontend

The whole frontend is composed out of 7 visual parts. On top of the page, there is a form for filtering. Below, there is a data table showing all fetched data, or depending on the tab that you choose, there are parts composed of graphs that visualise statistically analysed data. Other than visual parts, it contains two main React hooks that define the page's behaviour. There is a React hook responsible for data visualisation and a React hook responsible for the creation of a filter, which determines the data that will have been fetched. The whole visual look of the application can be seen in Figure 6.2, Figure 6.3, and Figure 6.1.

The first visual component—form—is composed of inputs for each value mentioned in database Subsection 6.2.1. It has 2 possible views, either extended for a full search of every value or reduced, which is more visually pleasing and it does not take a lot of space. There are also inputs labelled `"from"` and `"to"`, which define the time window in which we want to select flight disruptions. Some inputs take into account that there can be multiple inputs concatenated with the keyword `" or "`. When submitting, the frontend checks whether inputted data was written correctly. If so, it updates the filter React hook, which states which data will have been fetched in the future. After that, the asynchronous function creates a request that asks the application backend for data for visualisation. Upon receiving, it updates the React hook responsible for the visualisation of data. I used React hook as they trigger on-page re-rendering of affected components, resulting in updating graphs.

The second part of the website is the data table. It visualises raw data that was fetched from the application backend. The data table has an implemented button for each header column, which triggers sorting based on specific columns. The data table also provides a dropdown menu from which the user can define which columns he wants to see. Other than that, it contains a detailed view. Detail view is a button embedded into each row. Upon clicking it, all data are visualised into a more comprehendible form, and also, in case of email flight disruption, metadata such as sender of email, receiver of email and subject are shown. Lastly, the data table has pagination built in. As previously mentioned, visualising all data could break down the frontend. The data table has a pagination to visualise only part of the data. Upon demanding more data, the request is made for the application backend, demanding other rows of data. Upon receiving data, the data React hook is updated (with the filter React hook staying the same).

The third part of the website is all the graphs, which visualise the data. Especially there are 2 pie graphs for visualising disruption type and parsing source. One bar graph visualises the top carriers used. In the case of more than 30 carriers, the last n-30 carriers are united into one column. Then, a line chart visualises parsed compared to skipped compared to failed data over time, and lastly, there is a bar graph which visualises the time between the parsing of flight disruption and departure of the flight. It divides all flights into categories based on their parsing and departure time difference.

All these graphs are interactive. As for pie graphs, clicking on certain parts updates the filter React hook, and upon receiving data, using the data React hook, it updates the whole page using an updated filter with certain pie graph values. As for other graphs, the user can select the interval from which he/she wants to select data. The first click brings graphs to the state of selection. Moving the cursor will visualise which columns should be fetched. Upon clicking a second time, similarly as for pie graphs, the filter React hook is updated and shown data are refreshed using the React hook.

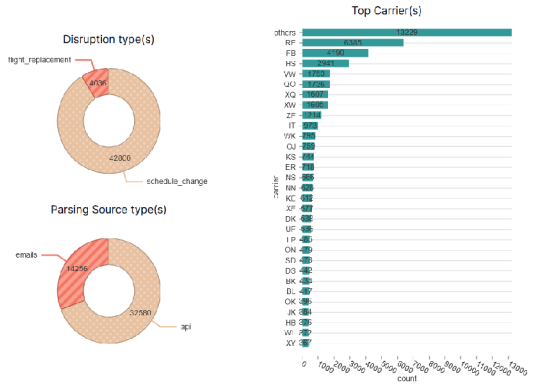

Figure 6.1: **Pie graphs and one bar graph of the application frontend.** Figure shows 2 pie graphs and one horizontal bar graph. Each of these graphs was made by aggregation of specific attributes of flight disruptions from Kiwi company. Either Disruption type, Parsing source, or Revised Segment Carrier. This is part of the second tab out of two tabs on the web frontend.

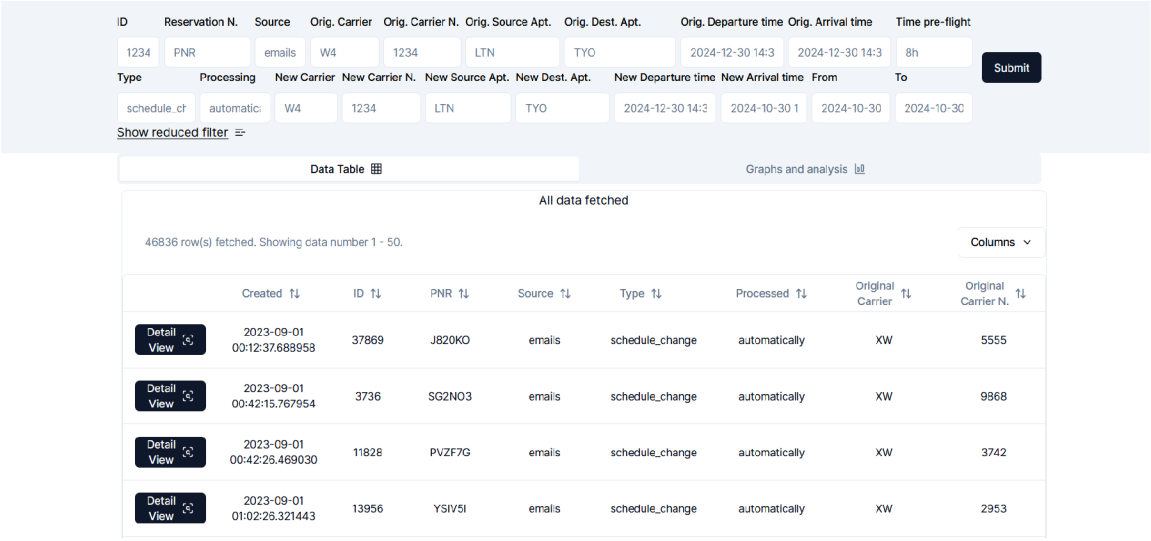

Figure 6.2: **Data table and extended filter of the application frontend.** Figure shows how was the data table visualised on the frontend side. It has key attributes which correspond to the values saved in the database. On top of the Figure, we can see the extended application form made for filtering of wanted values. This is part of the first tab out of two tabs on the web frontend.

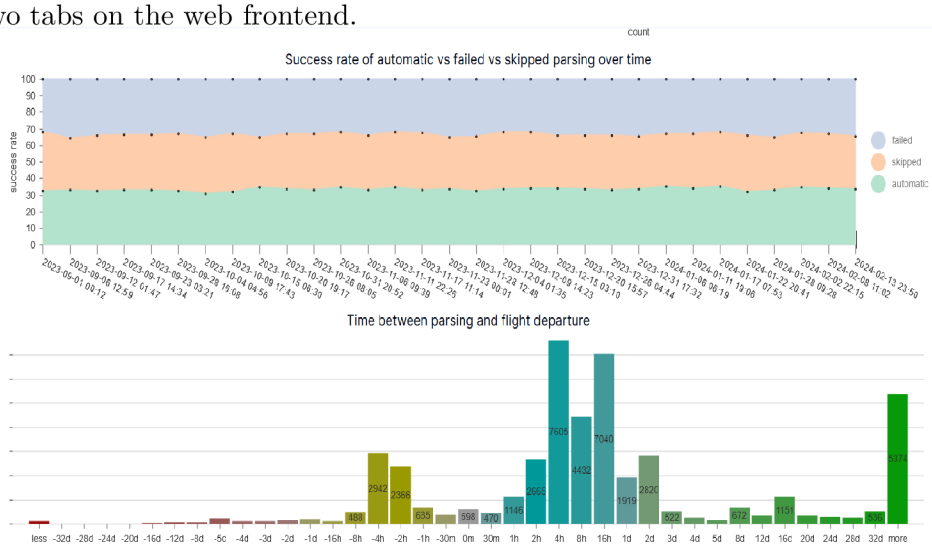

Figure 6.3: **Horizontal graphs of the web frontend.** Figure shows the visualisation of horizontal graphs. The first graph depicts the success rate over time. The second graph shows categories and the number of flight disruptions belonging to them. Each category represents how in advance the flight was parsed on the Kiwi side. This is part of the second tab out of two tabs on the web frontend.

Chapter 7

Testing of Application

Testing the whole application was performed in multiple steps. Firstly, during implementation. Every code change was followed by testing the whole application. Upon creating the whole application, extensive testing of the application and tuning were performed in order to create a better application. Secondly, after the application's functionality was achieved, testing was conducted with the help of other people or by using testing software. The aim was to find the correct solution that would be suitable for the Kiwi company. In the following sections, I describe the whole process of creating the tests and performing the analysis.

7.1 Unit Testing

As a part of the testing, I chose to perform unit tests. These tests should check whether the given parts of services work correctly. Seeing individual parts of the system perform correctly or incorrectly could pinpoint the weaknesses of a given code. The tests are performed on my application backend, and also, as an assurance that the app works correctly, I performed the tests on my Kiwi backend simulator. As both of these services are implemented in Python, I tested both of them using Pytest[1]. Upon completion of the tests, Pytest creates a user-friendly output with statistics of the tests. Therefore, I choose this software for testing. I performed testing on parsing parts of the application backend and Kiwi backend simulator.

The application backend parser checks the HTTP GET query that was received from the frontend. I simulated multiple queries that could be fetched from the frontend. For each filter, there was a query which checked whether the given filter refused incorrect value, whether the given filter accepted a correct value, and lastly, whether the given filter (if possible) accepted multiple values using the keyword `" or "`. This feature was described in Subsection 6.2.2. As some filters accept the same values, for example, the filter for original segment departure time and the filter for revised segment departure time both check for time, I tried to choose different values for each failed and parsed test. Then, the tests run through a more complex query, which uses multiple filters. Lastly, the tests check whether the output sent to the frontend is correct. Statistical analysis is performed on 50 flight disruptions, which the Kiwi company provided me. These tests include checking the dictionary, which is sent to the frontend for visualisation of raw data, checking statistical analysis of disruption types graph (aggregation by flight disruption type), source type graph

[1] https://docs.pytest.org/en/stable/index.html

(aggregation by flight disruption source), bar chart of carriers (aggregation by flight disruptions revised segment carriers), line chart for success rate (dividing of flight disruptions into time segments and calculating their parsing success rate), bar chart for times before flight (categorising flight disruptions into pre-set categories of pre-departure times). The whole application backend was described in Subsection 6.2.3.

The Kiwi backend parser works very similarly to the application backend parser. Therefore, the tests performed for the Kiwi backend parser are similar. Firstly, the tests check whether the received HTTP GET request is correct. Similarly, as for the previous part, I check whether the parser can recognize incorrect values, correct values, and correct multiple values using the `" or "` keyword. However, after the filtering of parameters comes different stages of testing. As the next step, the Kiwi backend parser creates an SQL query, which is then sent to the database. Therefore, in the tests which I performed, I checked whether the given queries were correct. Firstly, I tried to determine whether the SQL query was correct for no filters provided, and then I checked whether it could parse a single value from each filter. Lastly, I checked whether the parser could create a query for multiple filters that were provided.

7.2 End-to-end Testing

One way to test the whole application is by using the automatic existing testing software. The purpose of the tests was to create expected output based on input without any knowledge of architecture within the application. For this purpose, I choose Cypress[2] for testing the application. In my case, it is the end-to-end testing of the whole application. Cypress helps testing by navigating through HTML tags and performing code-defined actions. This way, people can automatically perform fast tests upon changing the code of their application. It is suitable for debugging as it takes real-time snapshots of the applications while real-time commands are executed. On top of that, Cypress can perform tests on multiple platforms, expanding test possibilities. Also, for further development, which could happen in the future, I choose automatic tests as they may be integrated into Git Continuous integration and continuous delivery/deployment. This means that simply by committing a change, a suite of automatic tests will run through your new commit, checking its functionality and setting it up for deployment. While implementing the tests, I followed 3 main steps:

- Set up the application state.
- Take an action which will create an environment within which I expect to have results.
- Check the resulting application state and make needed assertions.

The tests which I implemented check the basic and expected functionality of the application. The tests are divided into 3 main categories, starting from simpler tests to more complex testing. The first set of tests checks the form as an initializing element within the page. It checks for the existence of every filter that there is, the submit button, and the existence of the switch to the advanced filter. The tests do the same for advanced filter. Upon correct inputting of values, the code also tries to input incorrect values to HTML inputs to see whether the application will respond or not.

The second suit of test tries to use the basic functionality of a web page. Each test enters a single value into one input and checks the answer for fetched data. It is done using an advanced filter and also using the reduced filter. After this search, the tests try to fetch

[2] https://www.cypress.io/

all data and check whether fetched data withdrawn graphs have good values. The last part of this test suit is the test for pagination. The tests try to move through the pages of data back and forth. Each time it visits a certain page, the test expects there to be the same value.

The third suit of tests checks for graphs and their interactivity. Each graph within the page should be interactive. Therefore, the tests try selecting the data from the graphs. Firstly, the tests try to select only one instance from the graph (one table). After that, the graphs are tested by selecting multiple instances. As a final test, the code tries to find certain data only with the use of graphs. Then, it tries to find certain data by using a form and tests together.

7.3 Testing in Kiwi Company

As a part of testing, I conducted in-person testing with the Kiwi company employees who will use and perform tasks with the whole application and to whom I will install the whole software. In total, there were 8 people, of whom 5 were developers (2 senior software-developers, 2 medior software-developers, 1 junior software-developer), one team leader, team manager and a quality assurance supporter. The team was provided with a project and asked to install it and first test it independently. After that, I asked them to test the tasks that I had given them. The tasks which I gave them are as follows:

- Filter out specific Flight Disruption based on the provided ID and based on the provided reservation number. Try to inspect the properties of Flight Disruption whether you can read them.
- Then followed filtering multiple flight disruptions based on different attributes. Each user was given a specific time window or set of destination and origin airports or a set of carrier and carrier numbers. Try to inspect the data fetched and the graphs shown.
- Try to filter by each graph. First, try filtering by multiple values, then try to filter out only a single value. Try combining form filter inputs with the interactivity of graphs.

Overall, the functionality of the project was achieved, and the work served its purpose. The users could navigate through the pages, fetch data from the databases, filter the ones that they need, they could read graphs, and they found graph interactivity helpful. The requirements from Section 4.4, which the employees needed, were met. The employees could search for the particular reservations. The developers were satisfied with the graphs that would help them maintain their code. That includes the disruption type graph, parsing source graph, success rate graph, and graph comparing parsing time and departure time of flight disruption. The team leader and manager were pleased with the results, which may help them while creating business decisions. That includes previously mentioned graphs with graphs depicting the most used carriers. However, there were many suggestions regarding the design of the work, styling and overall web look. The whole conclusion of their report is as follows:

- Docker and starting of the whole application—one junior software developer and quality assurance supporter had trouble starting up the project. They tried to start up the project using *docker-compose up* command. After further investigation, we

together found out that they had services running on allocated ports, which blocked them from turning on certain services of the web application. After shutting down services which had allocated the ports, their project started up.

- Layout and visuals—the whole visual aspect of the page was not pleasing to them. They would rather find data and graphs separately. Also, the whole page was rendered on the whole screen. They would prefer an empty margin from the left and right side of the screen. They like that the filter has advanced and reduced selection as a part of the layout. On top of that, they found it useful that it sticks to the top of the screen. This way, the user does not have to scroll to the top of the screen to change the filter values. On the other side, the names of filters could be shorter. People will get used to them, so it would be better if they did not take up so much space.

- Data table—one of the main functionalities of the project was focusing on data display. The first action that testers took was looking at the data table. They disliked that data table cells were far apart, creating very wide objects, and there were too many white spaces in the data table. Wide objects resulted in the creation of a horizontal scroll bar, which was, in their opinion, not user-friendly. Also, an issue occurred when one user tried to click the Next Page button rapidly. As there were many requests for the backend, eventually, it froze and crashed the whole application.

- Detail view—the data table provided a special view to see the whole flight disruption. Upon entering, they found it kind of disorganized and not eye-pleasing. On top of that, the detail view button was not visually striking, so they could not tell that the given feature existed. Many visuals were unnecessary for a detailed view and could have been better styled using icons. However, they really liked the visual part of styling the previous flight compared to the new flight. They said that the previous flight could be crossed out as a little improvement to the detail view. As their suggestion, having rather short data table rows and a more detailed and styled Detail view was proposed.

- Graphs were viewed positively in the whole work. Users found it really pleasing that graphs are interactive. The team leader, team manager, and quality assurance supporter found them really useful for the purpose of creating statistics. Everyone could comprehend how data selection works, but there were a few misunderstandings about the line chart. The line chart did not seem that much intuitive, with users struggling to select a given time interval. They used to double-click one value, selecting a time window which started and ended at the same time.

7.3.1 Fixing Suggestions from Kiwi Company Employees

As a part of fixing suggestions from Kiwi company employees mentioned in the previous section, I implemented features that should help with total navigation within the web page.

- Docker and starting the whole application—I added a descriptive README.md file, which describes the process of starting up the whole service and possible problems which may be encountered along the way.

- Layout and visuals—I created two tabs on the page, showing different parts of the view. One tab shows the data table and all its attributes. The second tab shows every graph and statistical analysis of the view. On top of that, the whole page now has a

margin on the left and right sides, focusing the user's view on the centre of the page. As for filters, their names got shortened so they would not take up so much space. As a little improvement for overall graphics, I added a loading screen for fetching data through the submit form.

- Data table as an object got reduced significantly. Now, it is on a separate tab and does not have a horizontal or vertical scrollbar. Users can navigate through it by scrolling down the page. Also, most of the columns were deleted in order to shorten the rows in the table. More focus on detail was left in the detail view option of the row. As for pagination buttons, they are now disabled as the user waits for one page to load.

- Detail View as a function got highlighted within the whole page. Before the testing, the button within the ID of the row triggered the detail view. After testing, the detail view is triggered by a separate button within one row. As a little improvement, I crossed out previous flights as mentioned by Kiwi company employees. The order and alignment of components of the detail view were changed. Also, as a better visualisation of properties of the detail view, I added icons for some properties like sender, which suits to have an icon. Based on this improvement, icons were also added to other parts of the page.

After the testing in the Kiwi company and fixing their suggestions, I presented a new solution to them, which they found much more helpful. They found the newer solution more eye-pleasing and more organized than the previous solution.

Chapter 8

Conclusion

My goal was to create a web application that could help the Kiwi company employees perform daily tasks while using multiple databases filled with flight disruption data. The application visualises raw data along with statistically analysed data into graphs. Before implementing the web service itself, I researched flight disruptions, their cause and therefore also briefly air traffic. Then, I researched data visualisation and best practices, as well as how and when to visualise data. When I was done with my external research, I reached out to the Kiwi company to conclude an analysis of their current state of parsing flight disruptions and the current state of their software stack. Then, I created the design of a web application based on my research. Lastly, I implemented the provided application and performed the tests to ensure a better application run.

The web application was created specifically to visualise raw data and create a statistical analysis of key data attributes. Along with the web application, I implemented services which the application would use. That includes the application backend, which would perform all data analytic tasks and send correct values to the frontend for visualisation. To develop this work, I simulated Kiwi company side services locally. That includes Kiwi databases, which were filled with comma-separated values provided by the Kiwi company, and the Kiwi backend for databases. All these services were successfully created and can perform data trading among each other, with graphs interactively changing the visualised data.

The application frontend can filter and fetch data from the backend and visualise them. The application backend can accept requests and send parsed data back to the frontend. It can also communicate with the Kiwi backend and request data. Kiwi backend and Kiwi databases store data and can perform basic data fetching for application services.

After conducting testing and bug repairing, the Kiwi company accepted my application. Currently, it is in the process of being deployed to the Kiwi company. In the second quarter of 2024, it will be deployed for everyone to use. However, the application backend will need to have some minor changes that will define how it will communicate with the real-time Kiwi company services. I expect that it will overcome changes after some time, which I would gladly help with, as Kiwi company practices agile programming.

As for this work, I created my own services from scratch to complete a solution with 4 functional services. I also published my results at the Excel@FIT 2024 conference (submission number 20[1]), where I highlighted my solution and approach to the visualisation of flight disruptions for other people interested who could use my research or code in future for further development.

[1] https://excel.fit.vutbr.cz/sbornik/

Bibliography

[1] Abdi, M. R. and Sharma, S. Information system for flight disruption management. *International Journal of Information Management*. 2008, vol. 28, no. 2, p. 136–144. DOI: https://doi.org/10.1016/j.ijinfomgt.2008.01.006. ISSN 0268-4012. Available at: https://www.sciencedirect.com/science/article/pii/S026840120800008X.

[2] Ahituv, N., Igbaria, M. and Sella, A. V. The effects of time pressure and completeness of information on decision making. *Journal of management information systems*. Taylor & Francis. 1998, vol. 15, no. 2, p. 153–172.

[3] Borsky, S. and Unterberger, C. Bad weather and flight delays: The impact of sudden and slow onset weather events. *Economics of Transportation*. 2019, vol. 18, p. 10–26. DOI: https://doi.org/10.1016/j.ecotra.2019.02.002. ISSN 2212-0122. Available at: https://www.sciencedirect.com/science/article/pii/S2212012218300753.

[4] Breeden, A. French strikes disrupt traffic and flights. *International New York Times*. International Herald Tribune. 2016, p. NA–NA.

[5] ChartJs. *Chart.js* [online]. ChartJs, 2023 [cit. 2024-01-09]. Available at: https://www.chartjs.org/docs/latest/.

[6] D3. *What is D3?* [online]. D3, 2023 [cit. 2024-01-09]. Available at: https://d3js.org/what-is-d3.

[7] Delaune, E. F., Lucas, R. H. and Illig, P. In-flight medical events and aircraft diversions: one airline's experience. *Aviation, space, and environmental medicine*. Aerospace Medical Association. 2003, vol. 74, no. 1, p. 62–68.

[8] Dück, V., Ionescu, L., Kliewer, N. and Suhl, L. Increasing stability of crew and aircraft schedules. *Transportation Research Part C: Emerging Technologies*. 2012, vol. 20, no. 1, p. 47–61. DOI: https://doi.org/10.1016/j.trc.2011.02.009. ISSN 0968-090X. Special issue on Optimization in Public Transport+ISTT2011. Available at: https://www.sciencedirect.com/science/article/pii/S0968090X11000350.

[9] Enoch, J., McDonald, L., Jones, L., Jones, P. R. and Crabb, D. P. Evaluating whether sight is the most valued sense. *JAMA ophthalmology*. American Medical Association. 2019, vol. 137, no. 11, p. 1317–1320.

[10] Esirgapovich, K. A. et al. THE EASIEST RECOMMENDATIONS FOR CREATING A WEBSITE. *Galaxy International Interdisciplinary Research Journal*. 2022, vol. 10, no. 2, p. 758–761.

[11] Few, S. *Information dashboard design: The effective visual communication of data.* O'Reilly Media, Inc., 2006.

[12] Dwork, J. *What do the Lumo Delay Indexes mean?* [online]. Resilient Ops, Inc., 2018 [cit. 2023-12-28]. Available at: https://www.thinklumo.com/news/what-do-the-lumo-delay-indexes-mean.

[13] Google. *Using Google Charts* [online]. Google, 2023 [cit. 2024-01-09]. Available at: https://developers.google.com/chart/interactive/docs.

[14] Katunin, A., Dragan, K. and Dziendzikowski, M. Damage identification in aircraft composite structures: A case study using various non-destructive testing techniques. *Composite Structures.* 2015, vol. 127, p. 1–9. DOI: https://doi.org/10.1016/j.compstruct.2015.02.080. ISSN 0263-8223. Available at: https://www.sciencedirect.com/science/article/pii/S0263822315001683.

[15] Klapper, E. S. and Ruff Stahl, H.-J. K. Effects of the pilot shortage on the regional airline industry: A 2023 Forecast. *International Journal of Aviation, Aeronautics, and Aerospace.* 2019, vol. 6, no. 3, p. 2.

[16] Kohl, N., Larsen, A., Larsen, J., Ross, A. and Tiourine, S. Airline disruption management—Perspectives, experiences and outlook. *Journal of Air Transport Management.* 2007, vol. 13, no. 3, p. 149–162. DOI: https://doi.org/10.1016/j.jairtraman.2007.01.001. ISSN 0969-6997. Available at: https://www.sciencedirect.com/science/article/pii/S0969699707000038.

[17] Malandri, C., Mantecchini, L. and Reis, V. Aircraft turnaround and industrial actions: How ground handlers' strikes affect airport airside operational efficiency. *Journal of Air Transport Management.* 2019, vol. 78, p. 23–32. DOI: https://doi.org/10.1016/j.jairtraman.2019.04.007. ISSN 0969-6997. Available at: https://www.sciencedirect.com/science/article/pii/S0969699719300183.

[18] Midway, S. R. Principles of effective data visualization. *Patterns.* Elsevier. 2020, vol. 1, no. 9.

[19] Nadkarni, P. M., Ohno Machado, L. and Chapman, W. W. Natural language processing: an introduction. *Journal of the American Medical Informatics Association.* september 2011, vol. 18, no. 5, p. 544–551. DOI: 10.1136/amiajnl-2011-000464. ISSN 1067-5027. Available at: https://doi.org/10.1136/amiajnl-2011-000464.

[20] Nygren, E., Aleklett, K. and Höök, M. Aviation fuel and future oil production scenarios. *Energy Policy.* 2009, vol. 37, no. 10, p. 4003–4010. DOI: https://doi.org/10.1016/j.enpol.2009.04.048. ISSN 0301-4215. Carbon in Motion: Fuel Economy, Vehicle Use, and Other Factors affecting CO2 Emissions From Transport. Available at: https://www.sciencedirect.com/science/article/pii/S0301421509003152.

[21] Pfeiffer, M. B., Kougher, J. D. and DeVault, T. L. Civil airports from a landscape perspective: A multi-scale approach with implications for reducing bird strikes. *Landscape and Urban Planning.* 2018, vol. 179, p. 38–45. DOI:

https://doi.org/10.1016/j.landurbplan.2018.07.004. ISSN 0169-2046. Available at: https://www.sciencedirect.com/science/article/pii/S0169204618305917.

[22] Rupp, N. G. and Holmes, G. M. An investigation into the determinants of flight cancellations. *Economica*. Wiley Online Library. 2006, vol. 73, no. 292, p. 749–783.

[23] Su, Y., Xie, K., Wang, H., Liang, Z., Art Chaovalitwongse, W. et al. Airline Disruption Management: A Review of Models and Solution Methods. *Engineering*. 2021, vol. 7, no. 4, p. 435–447. DOI: https://doi.org/10.1016/j.eng.2020.08.021. ISSN 2095-8099. Available at: https://www.sciencedirect.com/science/article/pii/S2095809921000175.

[24] Suzumura, T., Kanezashi, H., Dholakia, M., Ishii, E., Napagao, S. A. et al. The Impact of COVID-19 on Flight Networks. In: *2020 IEEE International Conference on Big Data (Big Data)*. 2020, p. 2443–2452. DOI: 10.1109/BigData50022.2020.9378218.

[25] Talreja, R. and Phan, N. Assessment of damage tolerance approaches for composite aircraft with focus on barely visible impact damage. *Composite Structures*. Elsevier. 2019, vol. 219, p. 1–7.

[26] Tonidandel, S., King, E. B. and Cortina, J. M. *Big data at work: The data science revolution and organizational psychology*. Routledge, 2015.

[27] Tufte, E. R. *The visual display of quantitative information*. Graphics press Cheshire, CT, 2001.

[28] Ramotion. *Website Data Visualization Techniques That Make Information Understandable* [online]. Cloudinary, 2023 [cit. 2024-01-09]. Available at: https://cloudinary.com/guides/front-end-development/front-end-development-the-complete-guide.