



Semester Project

Cloud Computing

Tatjana Afanasjeva 273445
Wirginia Szoltysek 132841
Patricia Poracova 251818

Supervisors:

Richard Brooks
Jakob Knop Rasmussen
Laurits Ivar Anesen

24 424 characters

VIA Engineering

6th semester

17.12.2020

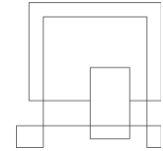
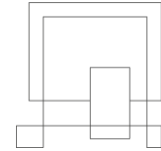


Table of content

Abstract	3
Introduction	4
Analysis	5
Functional requirements	5
Use Case diagram	6
Activity diagram	7
Design	8
System Architecture	8
Database ER model	9
User interface	11
Sequence diagram	13
Implementation	13
API structure	13
User interface structure	15
Results and Discussion	17
Conclusions	18
Project future	19
Sources of information	20
Appendices	20
Appendix A - Cloud User Guide	20
Appendix B - Source code links and Web application link	20
Appendix C - Sequence diagram	20



Abstract

The purpose of the project is to create an application that is hosted and delivered through a cloud computing platform and can be accessed remotely in order to visualize the "United Airplanes Association" desired data.

Development of the system is performed as an iterative process that includes the following steps - requirements, analysis, design, implementation, deployment, and testing.

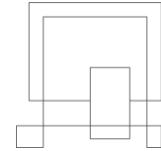
Requirements, use case diagram, and activity diagram were defined in the analysis part. System architecture, and user interface design are described in the design part.

Project structure, applied methods and frameworks are explained in the implementation part.

The system is implemented using JavaScript programming language, MySQL relational database, Node.js was used as a JavaScript runtime environment, Express.js web application framework was used to create an REST API endpoints, and Vue.js framework together with Apex chart library to build a user interface.

Microsoft Azure cloud computing services are used to host the system. To enable continuous integration and continuous deployment to the cloud, the GitHub action pipeline was used.

The current result is that the user can access the web application <https://salmon-dune-0b8d2b703.azurestaticapps.net/> through the internet and see all requested data visualizations.



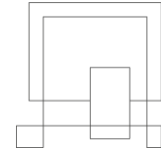
Introduction

The United Airplanes Association expands digitalization of the company and invests resources into faster and better fitted online applications. One of the areas for future development is visualizing airport-related data like temperature at the airport or mean departure delay etc. Selected managers of the company are required to have constant access to the data, therefore it should be available from any place with internet connection. The association decided it should be displayed through the web browser on managers' tablets and it should not be hosted on company own premises.

The purpose of the project is to examine the concept of Cloud Computing and the strategic considerations for using Cloud Computing based on development of data visualization system.

The main challenge of the project is to research available cloud providers and devOps tools for agile application development, choose the most fitting one and to implement it with the task of displaying data connected to airlines graphically to a user who is provided only with the url.

The report begins with requirements based on the interview with the United Airplanes Association. Next in the Analysis section, those requirements are used to create a domain of the problem and use case diagrams. Subsequently, design and implementation parts are presented. The last chapters contain testing, summary of the results of the project and conclusion.

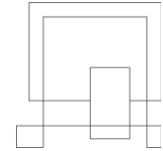


Analysis

Functional requirements

System must be able to display/visualize following data:

1. Total number of flights per month.
2. Total number of flights per month from the three origins in one plot including
 - Frequency,
 - Stacked frequency,
 - Stacked percentage.
3. The top-10 destinations and how many flights were made to these from all three origins.
4. The number of flights from the three origins to the top-10 destination.
5. The mean airtime of each of the origins in a table.
6. How many weather observations there are for the origins in a table.
7. For each of the three origins, all temperature attributes (i.e. attributes involving a temperature unit) in degree Celsius.
8. The temperature (in Celsius) at JFK.
9. The daily mean temperature (in Celsius) at JFK.
10. The daily mean temperature (in Celsius) for each origin in the same plot.
11. Mean departure and arrival delay for each origin in a table.
12. The manufacturers that have more than 200 planes.
13. The number of flights each manufacturer with more than 200 planes are responsible for.
14. The number of planes of each Airbus Model.



Use Case diagram

Figure 1 below shows the main users' interactions with the system. Based on requirements are identified three use cases.

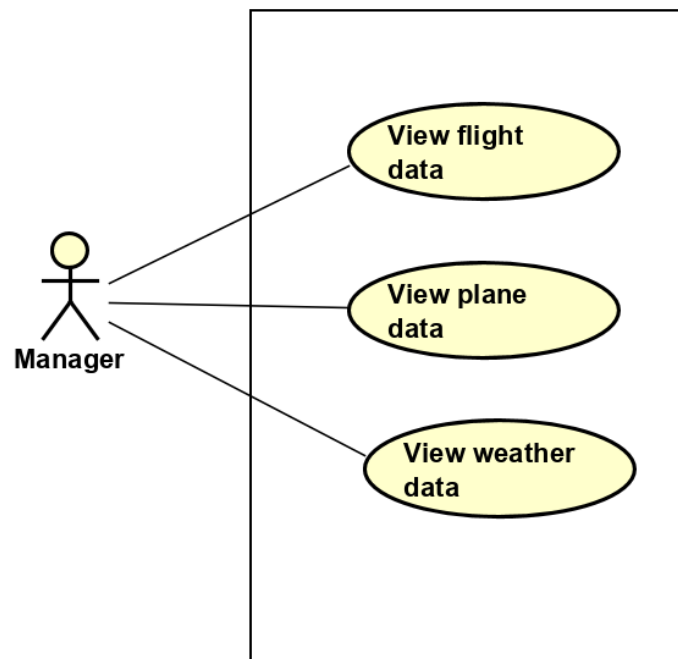


Figure 1 Use case diagram



Activity diagram

Activity diagram(Figure 2) represents the system's dynamic behavior when the manager opens the website with requested United Airplanes Association data.

When the user enters the website address, the system must return all requested data.

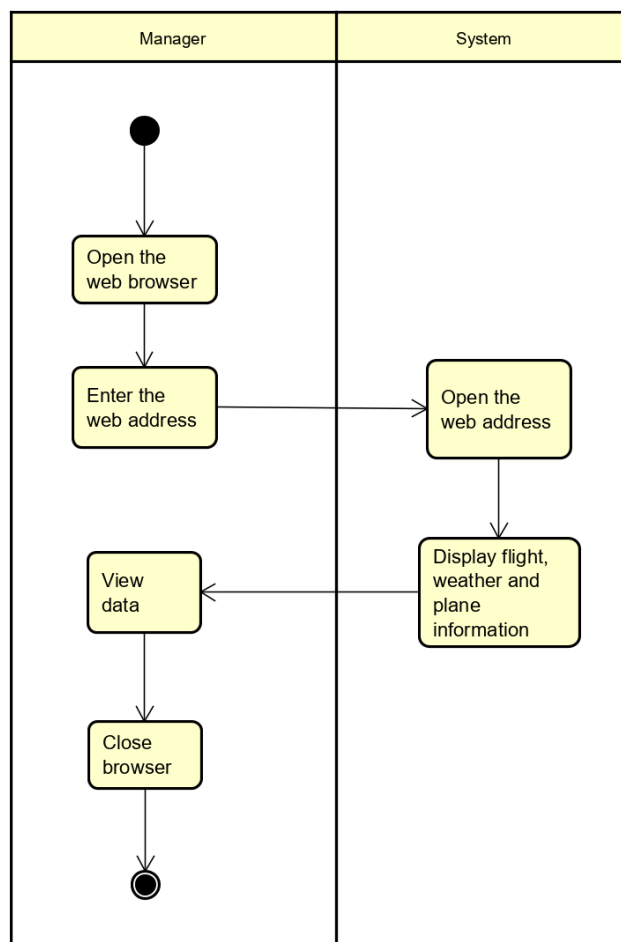


Figure 2 Activity diagram



Design

System Architecture

System will be built with the following architecture (Figure 3).

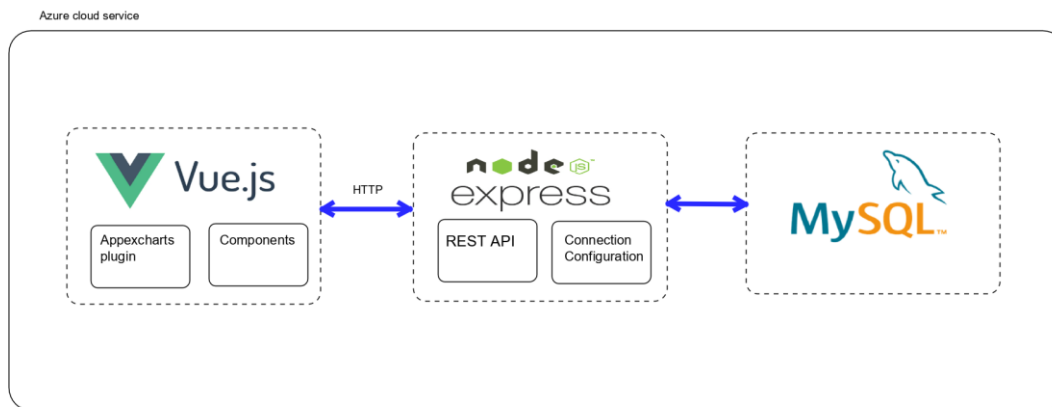


Figure 3 System Architecture

Node.js express server creates RESTful APIs which connects to MySQL server with the relational database. To create a user interface Vue.js (JavaScript framework) will be used. It will be responsible for consuming those APIs. To visualize data the JavaScript, library Appexcharts, which is a charting library that helps creating interactive and responsive charts and graphs, will be used.

The system will be deployed using Azure cloud services at an early stage of development (See Appendix A).



Database ER model

Data given from the customer in plain text file stored in the CSV files are transferred to the relational database, to ease the process of fetching the exact information which is requested in requirements.

Figure 4 below represents a relational model of “The United Airplanes Association” database with 5 tables.

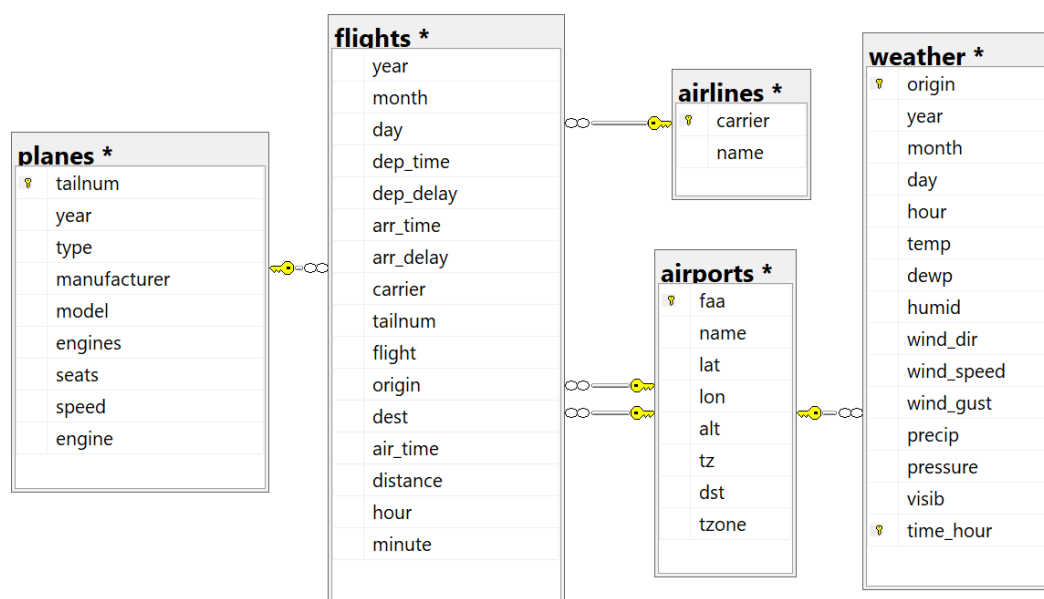


Figure 4 ER diagram



REST API

Every URI acts like a resource, and since it is only necessary to read data from the database, all the API methods are GET methods. The following Table 1 represents URIs that will fetch data from the database.

Table 1 API endpoints

Method	URL	Returned data
GET	/flights	Total number of flights per month
GET	/flights/origin	Total number of flights per month from the three origins in one plot
GET	/flights/origin/airtime	The mean airtime of each of the origins in a table
GET	/flights//top-10-destinations	The top-10 destinations and how many flights were made to these from all three origins.
GET	/flights/origin/mean-delay	Mean departure and arrival delay for each origin in a table
GET	/planes/manufactures	The manufacturers that have more than 200 planes
GET	/planes/manufactures/airbus	The number of planes of each Airbus Model
GET	/planes/manufactures/res	The number of flights each manufacturer with more than 200 planes are responsible for
GET	weathre/origin	How many weather observations there are for the origins in a table
GET	weather/origin/mean-temperature	The daily mean temperature (in Celsius) for each origin in the same plot
GET	weather/temperature	For each of the three origins, all temperature attributes in degree Celsius



User interface

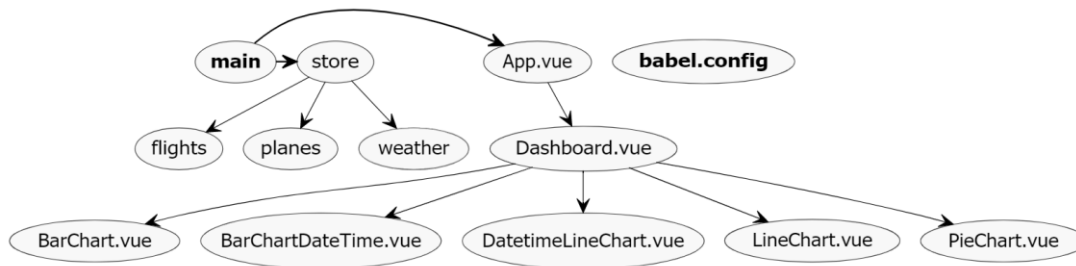


Figure 5 UI Class diagram

This is the basic class diagram (Figure 5) which will be used developing the user interface. Where the store is a state management system (controller + model) and Dashboard is View.

Based on the activity diagram, the design of the web page was developed (Figure 6, 7), where the user can see all the requested information when opening the web address.

By scrolling the page up and down, users can view flight, weather, and aircraft data in various charts.

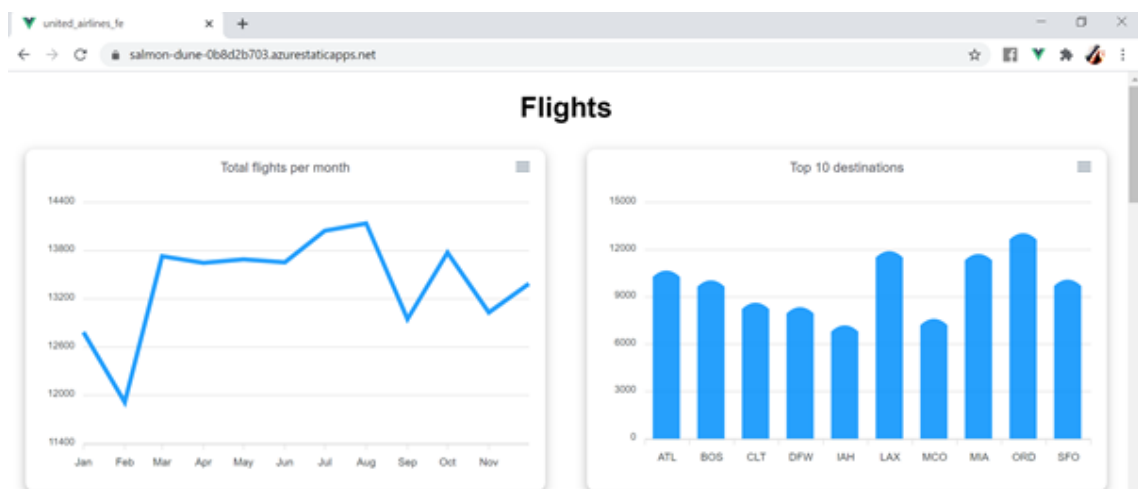


Figure 6 UI design_1

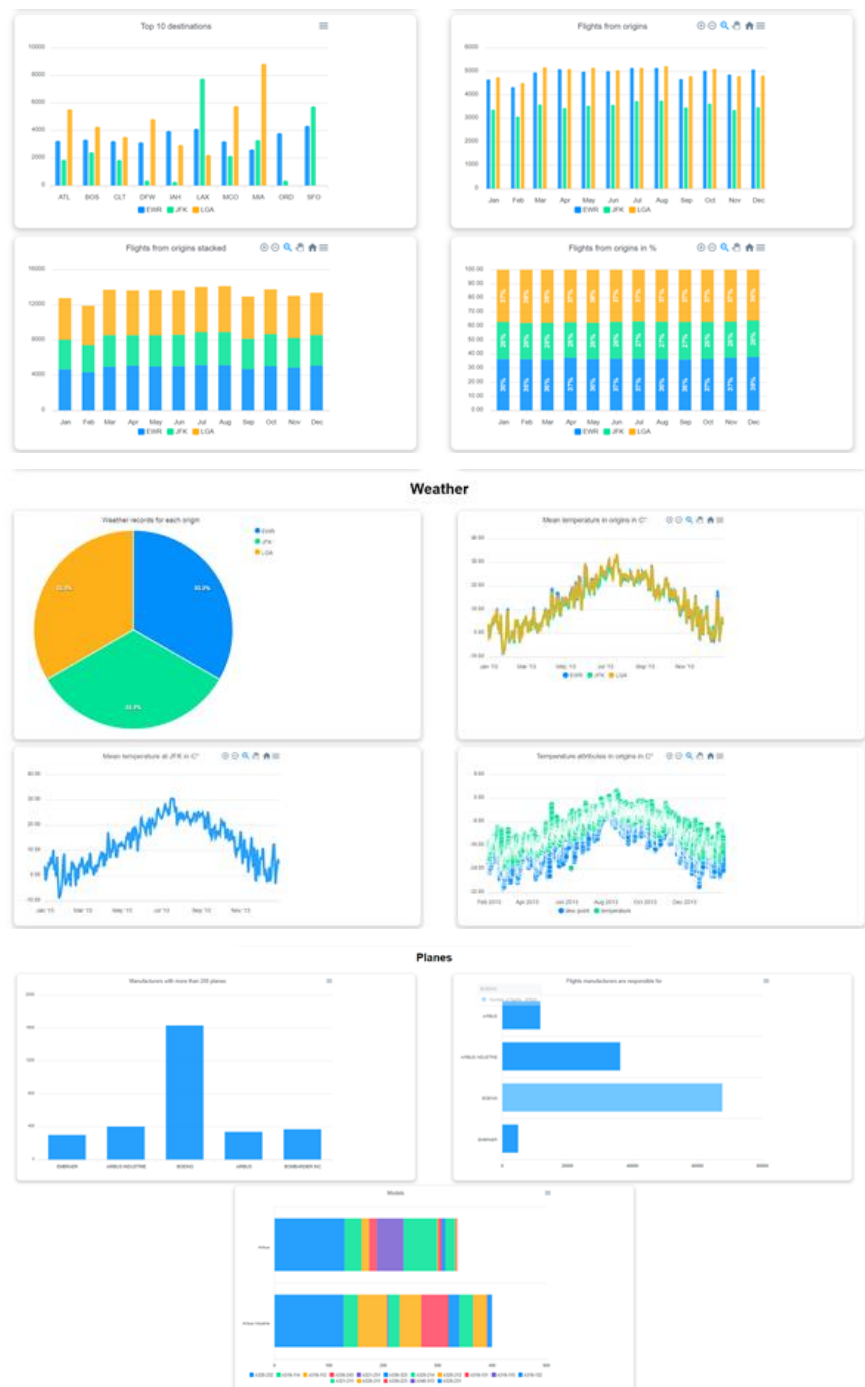
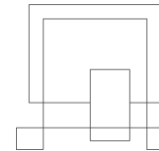
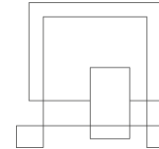


Figure 7 UI design_2



Sequence diagram

Sequence diagram below (Figure 8) shows how flight data are displayed to the user.

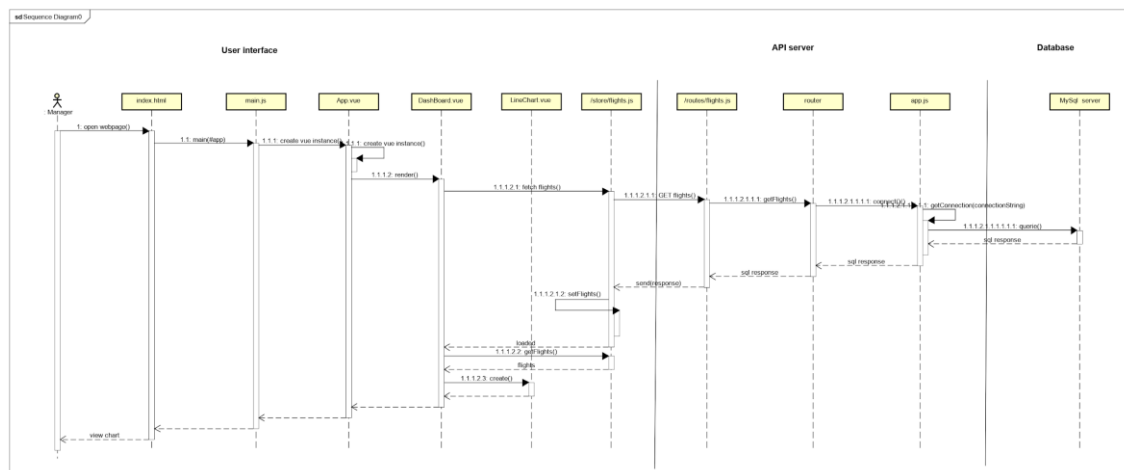


Figure 8 Sequence diagram

See diagram in Appendix C

Implementation

API structure

Figure 9 below shows the structure of API.

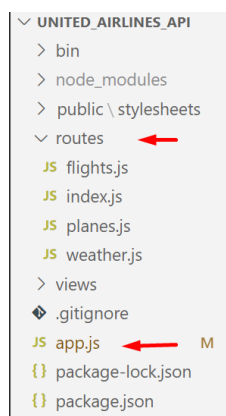


Figure 9 API structure



package.json holds all the dependencies required for the project.

app.js is the file that will be executed first in this node.js project. The declaration for the MySQL database connection is established in this file(See figure 10).

```
const mysql = require('mysql');
const myConnection = require("express-myconnection");

let config =
{
  host: "sep6db.mysql.database.azure.com",
  user: "sep6@sep6db",
  password: "5epsix1234",
  database: 'united_airplanes_db',
  ssl: true
};

const app = express();

app.use(myConnection(mysql, config, 'single'));
app.use(express.json());
```

Figure 10 Database connection

Having a connection string exposed as a part of the code stored on GitHub is not safe anyhow.(See chapter: discussions)

The main routes (flights, weather, planes) are defined in a app.js file. These routes are chained together with routers as shown in figure 11.

```
let indexRouter = require('./routes/index');
let flightsRouter = require('./routes/flights');
let planesRouter = require('./routes/planes');
let weatherRouter = require('./routes/weather');

app.use(myConnection(mysql, config, 'single'));
app.use(express.json());

app.use('/', indexRouter);
app.use('/flights', flightsRouter);
app.use('/weather', weatherRouter);
app.use('/planes', planesRouter);
```

Figure 11 Routes and routers



Files in routes folder defines API endpoints and reads data from the database as shown in Figure 12.

```
/*Total number of flights per month*/
router.get('/', function (req, res) {
  req.getConnection(function (err, connection) {
    let q = "SELECT month, count(*) as 'number_of_flights'"+
            "from `flights` group by month having COUNT(*)>=1 order by month";
    connection.query(q, function (error, results) {
      if (error) throw error;
      res.setHeader('content-type', 'application/json');
      res.send(results);
    });
  });
});
```

Figure 12 URI get total No of flights

User interface structure

Figure 13 shows frontend applications structure.

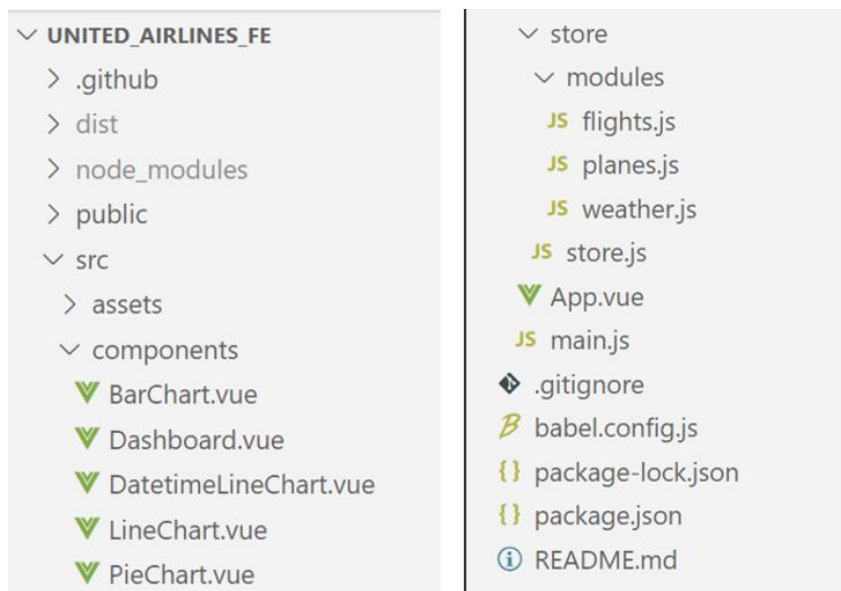


Figure 13 Vue.js frontend structure



The five different charts (bar chart, line chart, pie chart, bar chart with datetime x axis and line chart with datetime on x axis), were created as reusable components. These components are called in Dashboard.vue component, where data is passed into their props as seen in the code below (Figure 14).

```
<h1>Flights</h1>
<!--Total flights for each month-->
<lineChart chart-title="Total flights per month"
  :categories=this.months
  :series="[
    {
      name: 'Likes',
      data: this.flights
    }
  ]">
</lineChart>
```

Figure 14 Data passing to props

Data from the API is fetched in the application's store folder where each module is responsible to fetch different kinds of data(flights, weather, plane data). Every module has an action handler that contains asynchronous operations such as `fetchFlights` in Figure 15.

```
const kBaseUrl = "https://unitedairlinesassociation.azurewebsites.net/";
const actions = {
  fetchFlights({commit}) {
    axios.get(`${kBaseUrl}flights/`).then(response => {
      if (response.status === 200) {
        commit('SET_FLIGHTS', response.data);
      }
    })
  },
}
```

Figure 15 fetchFlights

These actions are triggered later in Dashboard.vue mounted lifecycle (meaning the page load) with the store.dispatch method as shown in the code below (Figure 16).



```
export default {
  name: 'Dashboard',
  mounted() {
    this.$store.dispatch('fetchFlights');
    this.$store.dispatch('fetchAirtime');
    this.$store.dispatch('fetchDelays');
    this.$store.dispatch('fetchTopTenDestinations');
    this.$store.dispatch('fetchOriginFlights');
  }
}
```

Figure 16 Dispatch actions

See the whole source code in Appendix B.

Results and Discussion

Table 2 Results

No	Requirement	Implementation
1.	Display the total number of flights per month.	Implemented
2.	Display the total number of flights per month from the three origins in one plot including frequency, stacked frequency, stacked percentage.	Implemented
3.	Display the top-10 destinations and how many flights were made to these from all three origins.	Implemented
4.	Display the number of flights from the three origins to the top-10 destination.	Implemented
5.	Display the mean airtime of each of the origins in a table.	Implemented
6.	Display how many weather observations there are for the origins in a table.	Implemented
7.	For each of the three origins display all temperature attributes (i.e. attributes involving a temperature unit) in degree Celsius.	Implemented
8.	Display the temperature (in Celsius) at JFK.	Implemented
9.	Display the daily mean temperature (in Celsius) at JFK.	Implemented
10.	Display the daily mean temperature (in Celsius) for each origin in the same plot.	Implemented



11.	Display mean departure and arrival delay for each origin in a table.	Implemented
12.	Display the manufacturers that have more than 200 planes.	Implemented
13.	Display the number of flights each manufacturer with more than 200 planes are responsible for.	Implemented
14.	Display the number of planes of each Airbus Model.	Implemented

Even though all requirements were fulfilled the implementation of the project is not finished. The APIs are lacking security, the connection string is exposed to anyone reading through our GitLab repository. The login for the front end as well as loaders when graphs are not ready to be displayed would help secure the webpage and improve user experience. The UI design of the implementation would also add a value, for now the app serves only as a tool for viewing the information.

See application in <https://salmon-dune-0b8d2b703.azurestaticapps.net/>

Conclusions

The report starts with requirements which contain a description of desired features from the perspective of future system users: managers of United Airplanes Association.

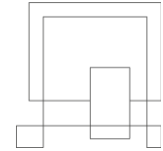
In the analysis part, the emphasis was put on the "View Flight data" use case and its main features.

Subsequently, the design part includes descriptions of system architecture, sequence diagram and user interface design. As well design parts contain a description of the database, ER diagram.

Sections Test and Results summarize which features were implemented and tested.

Application fulfils all requirements. It allows the manager to see all desired information about flights, weather and planes. All mentioned functionalities passed the end-user test.

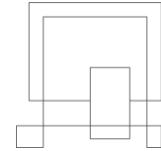
The security of API is not implemented, however that does not affect the main purpose of the project and will be left for future development.



Project future

Possible future improvements that could upgrade the system:

- securing API.
- login
- decreasing load time
- improving design aspect
- using Azure vault for storing environmental variables
- changing URL's name according to the United airplanes association's choice.



Sources of information

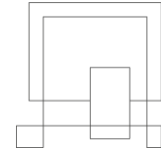
1. Vuejs.org. 2020. *Vue.Js*. [online] Available at: <<https://vuejs.org/>> [Accessed 16 December 2020].
2. ApexCharts.js. 2020. *Vue-Apexchart - A Vue Chart Wrapper For Apexcharts.Js*. [online] Available at: <<https://apexcharts.com/docs/vue-charts/>> [Accessed 16 December 2020].
3. Azure.microsoft.com. 2020. *Cloud Computing Services | Microsoft Azure*. [online] Available at: <<https://azure.microsoft.com/en-us/>> [Accessed 17 December 2020].
4. Google Cloud. 2020. *Cloud Computing Services | Google Cloud*. [online] Available at: <<https://cloud.google.com/>> [Accessed 17 December 2020].
5. Docker. 2020. *Empowering App Development For Developers | Docker*. [online] Available at: <<https://www.docker.com/>> [Accessed 17 December 2020].
6. Group 2 VIA ICT Project Report 20-12-2019/ Smart Cellar, 4th semester

Appendices

Appendix A - Cloud User Guide

Appendix B - Source code links and Web application link

Appendix C - Sequence diagram



Appendix A

Cloud User Guide

Tatjana Afanasjeva 273445
Wirginia Szoltysek 132841
Patricia Poracova 251818

Supervisors:

Richard Brooks
Jakob Knop Rasmussen
Laurits Ivar Anesen

VIA Engineering

6th semester

17.12.2020

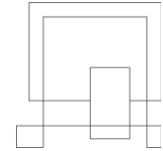


Table of content

Appendix A - Cloud User Guide	3
Cloud provider choice	3
MySQL server	5
Wedb Service	7
Static web app	9
Project management	9
Version control	14



Appendix A - Cloud User Guide

Cloud provider choice

The cloud provider of our choice was firstly google cloud, we have managed to create sql instance, connect it to our Node.js server and also create [cloudbuild.yaml](#) file to automate our deployment through version control, to be exact, in our case GitHub. For creating sql instances we followed the provided guide on Itslearning. We created this small skeleton too soon (2nd November week) and by the time we started working on a project fully we ran out of credit. Which is why we will not explain how we set up the google cloud in more detail. In the following screenshot you can see our cloudbuild.yaml (Figure 1) which is the only interesting part from this chapter of our project development. We used the steps field in the build config file to specify a build step.

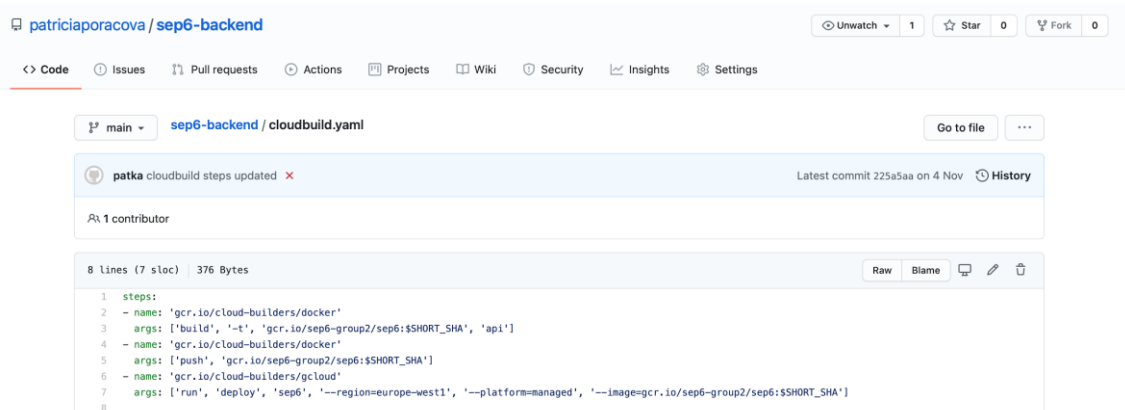


Figure 1 cloudbuild.yaml file

The `name` field of a build step to specify a [cloud builder](#), which is a container image and we called docker build. The `args` field of a build step takes a list of arguments and passes them to the builder referenced by the `name` field. So in the first step we build our docker container, then we push the container to the cloud and lastly we run our container. We were mostly just following google cloud documentation and got our hands on the console, cloud build and cloud run and services which we needed for setting this database and server up.



Following screenshots (Figure 2) showcases our cloud builds which were triggered by GitHub push to master branch.

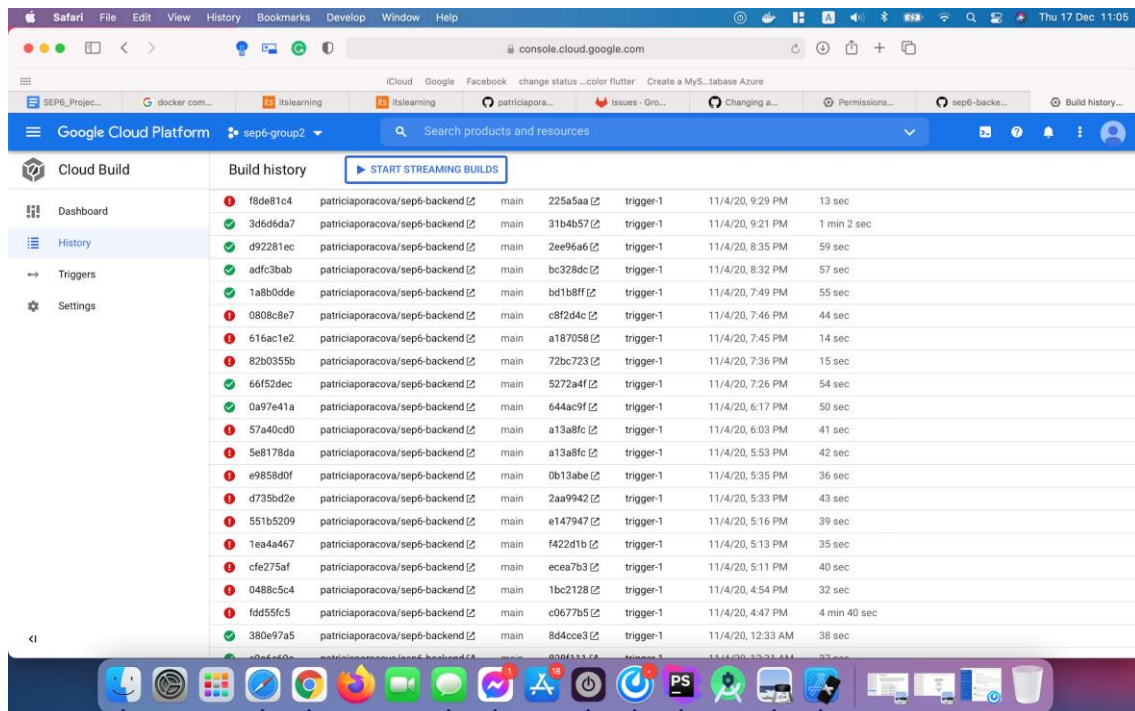
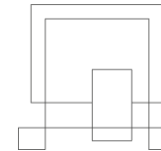


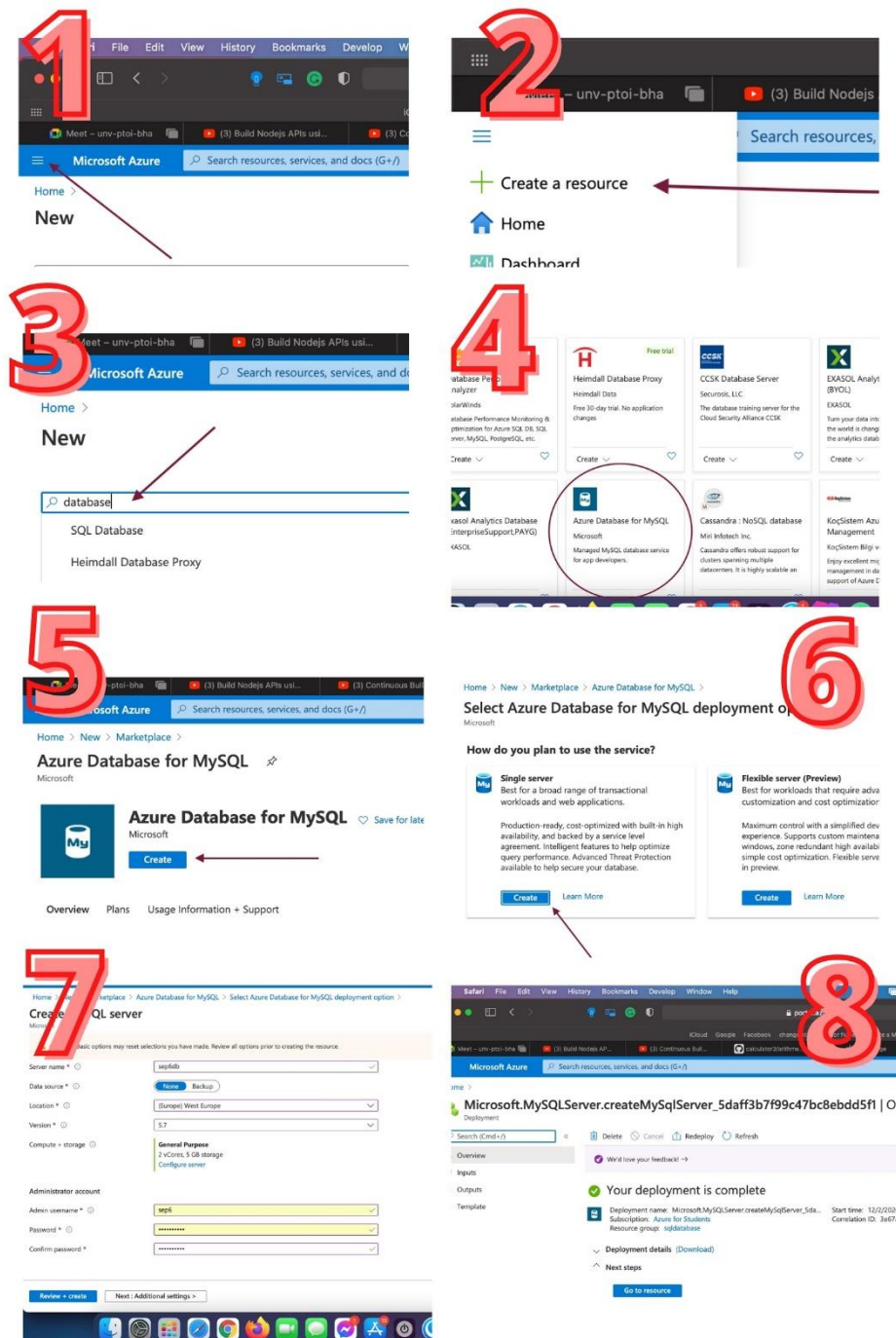
Figure 2 cloud build

As mentioned above we run out of credit before completing the project so we decided to change the provider for Microsoft azure. In the next subchapter we will describe and provide screenshots on how did we deployed all three of our parts.

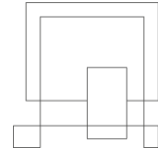


MySQL server

Figure 3
MySQL



database creation



After creating MySQL database, we tried to connect to it via PhpStorm and were met with expected error (Figure 4) which was “client with IP address xx.xxx.xx.xxx is not allowed to connect to this MySQL server.”

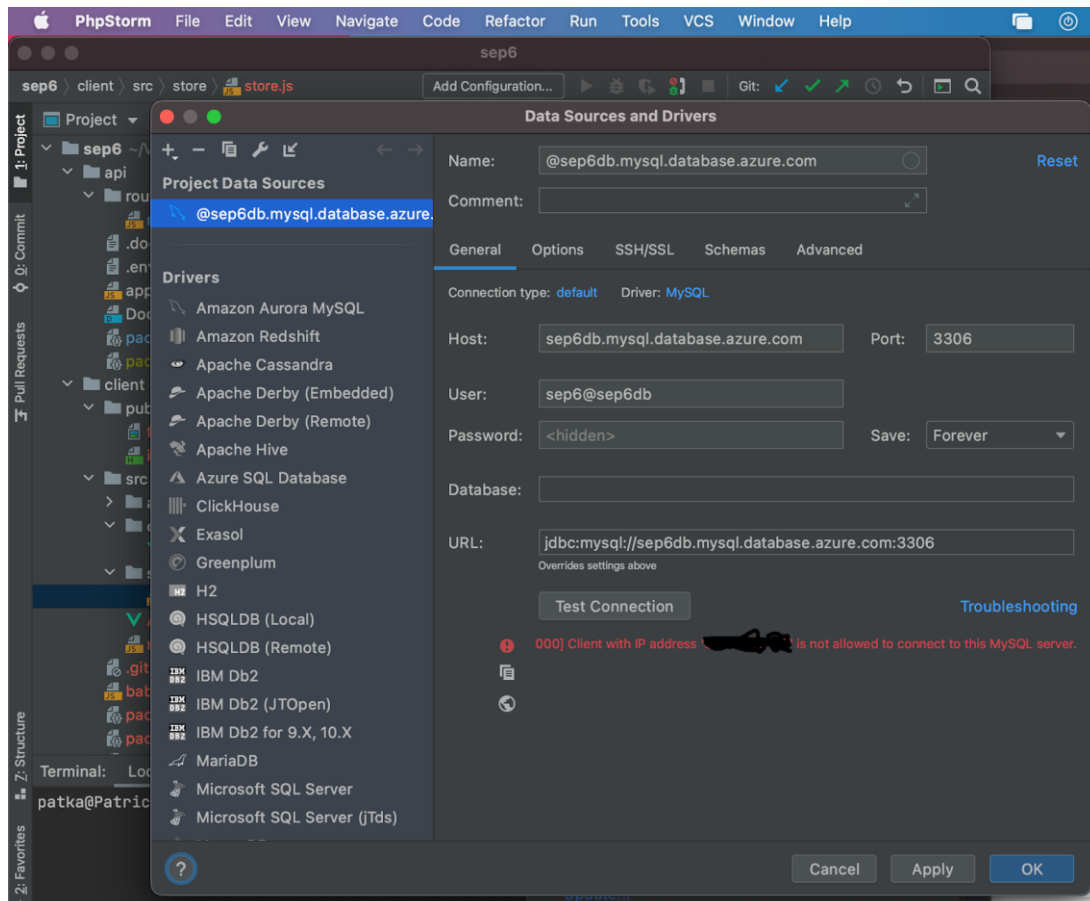
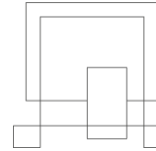


Figure 4 IP error

This was solved with adding an IP client to our server in the security tab. This had to be done for each one of us once we moved from developing locally through docker to having our services up in the cloud. More about our local development can be found in chapter project management.



Web Service

We followed almost the same steps as in making the MySQL server, that means, creating a new resource, looking for a Web App. The following steps can be seen on figure 5.

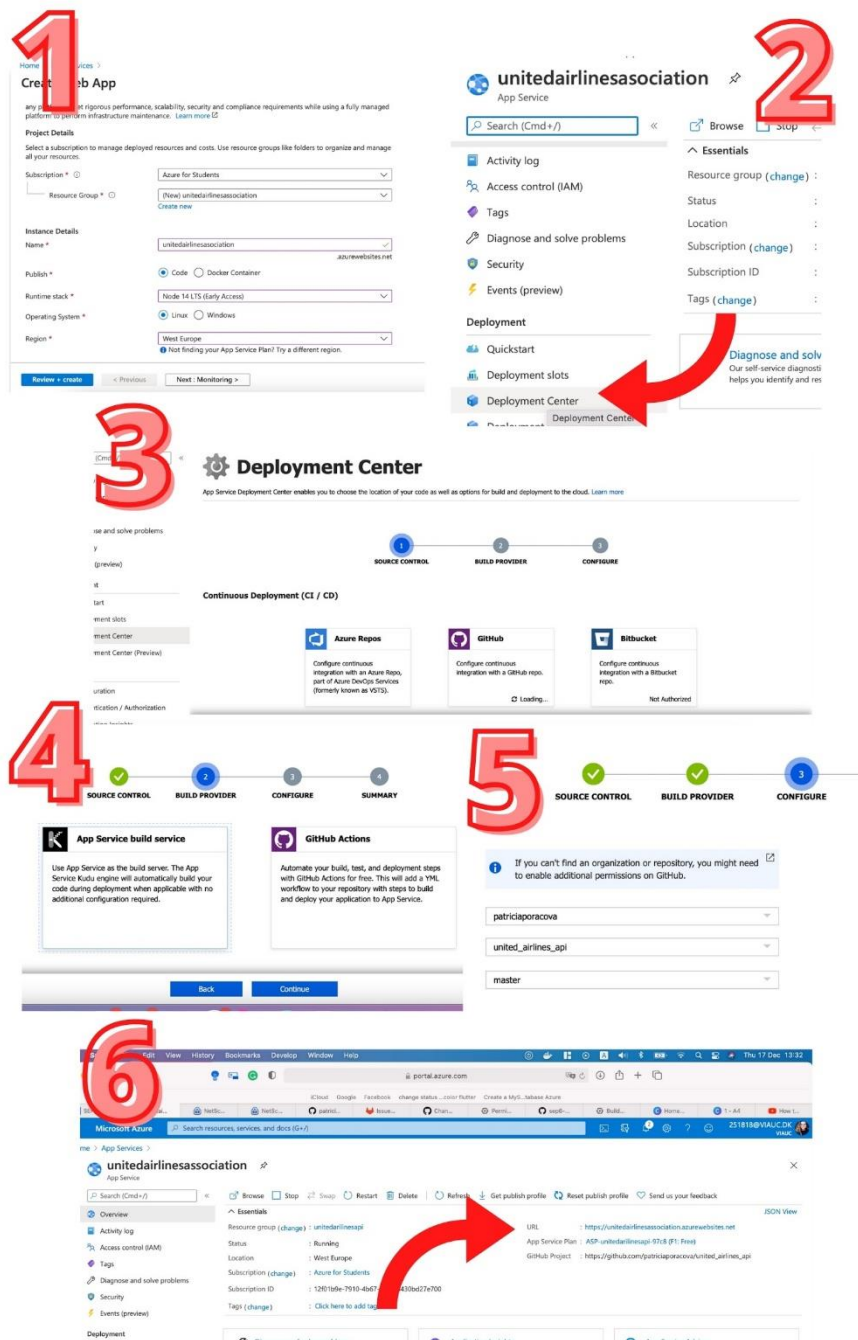
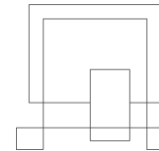


Figure 5 Create Web Service



We filled in the form, specifying resource group, name of the instance, then if we want to publish code or container to which we have chosen code and run time stack which was in our case Node 14. When the Web App was created, we went to the tab deployment center to hook it with our already existing GitHub repository. We specified the build provider and chose the right repository and then we were all set up. The portal redirected us to an overview of our web app where we saw the URL of our web service.

More tricky part was to connect to our database. We had to go to tab configuration and add a connection string with the content that can be seen in following screenshot (Figure 6).

Add/Edit connection string

Name: MS_TableConnectionString

Value: Data Source=tcp:sep6db.mysql.database.azure.com, 3306; Initial Catalog=sep6db; User ID=sep6@sep6db; Password = Sepsix1234

Type: MySQL

☐ Deployment slot setting

Figure 6 Connection string

The obstacle we were facing with our web service was securing the connection string in node.js part. This was supposed to be achieved by `process.env.ENV_VARIABLE`, seeing the build log (Figure 7) you can see we spend one night with it without luck, we then decided to leave it for later, but did not have time to try again.

Sunday, December 6, 2020				
1:37:56 AM GMT+1	Success	206d294 (patka)	updating package.json	
1:35:55 AM GMT+1	Success	51d9a9e (patka)	enabling CORS	
1:11:19 AM GMT+1	Success	f8a3625 (patka)	sql config fix	
12:45:27 AM GMT+1	Success	528e3c6 (patka)	sql connection fix	
12:27:25 AM GMT+1	Success	e578910 (patka)	sql connection string updated 3.0	
12:19:19 AM GMT+1	Success	d7a527d (patka)	sql connection string updated 2.0	
12:03:50 AM GMT+1	Success	728ff8b (patka)	sql connection string updated	
Saturday, December 5, 2020				
10:56:56 PM GMT+1	Success	d24e610 (patka)	sql connection	

Figure 7 Env variable



Static web app

Setting up the front end contained the same steps the only difference was to choose Static web app instead of web app. We were struggling with changing the web URL. We tried to follow a guide which suggested creating DNS zone (Figure 8), but there were some errors on our side and we also decided that it is not a priority in our project.










Name	Type	Last Viewed
 unitedairlinesassociation	App Service	4 minutes ago
 unitedairlinesassociation	Resource group	7 hours ago
 unitedairlines	Static Web App (Preview)	a week ago
 unitedairlinesassociation.com	DNS zone	a week ago
 azuredsn	Resource group	a week ago
 unitedairlinesapi	Resource group	a week ago
 sep6db	Azure Database for MySQL server	a week ago
 unitedairlines	Resource group	2 weeks ago
 sqldatabase	Resource group	2 weeks ago

Figure 8

Project management

Because the development of the project was a bit hectic from the beginning, we actually ended up with two different working repositories. We managed to set up our local developments using docker-compose.yaml file, then we had failed with google cloud and then finally we were developing in iterations using version control and CI/CD pipelines, deploying our code 'straight' to cloud.

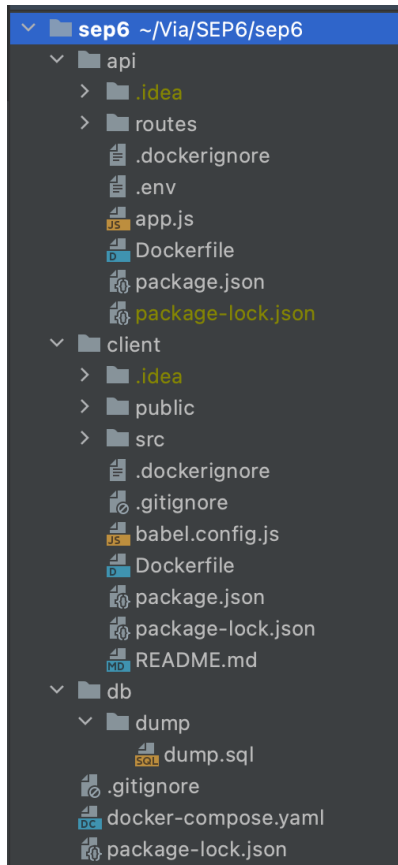


Figure 9 folder structure

To explain each phase more in detail, after receiving a list requirement we decided to do REST API in Node.js connected to MySQL database and Vue.js front end. We therefore created a folder with the following structure (Figure 9).

Main folder sep6 held the 3 parts of our project: client, api and db in which we had the sql dumb file.

The following figure 10 shows the content of our docker-compose.yml file. We used version 3 of the compose file, and our services (containers) were MySQL, api-server and client. MySQL used official MySQL 5.7 image available on docker hub. We specified the port and environmental variables used to be able to connect to our database. Volumes are used for data persistence meaning that when we modified a database and stopped or restarted our container the data modifications would be still there. The

second line of volumes key is there for initial data load. Since we were given data, we had to fill up our database on initialization.

Both client and api-server have their own docker files which are specifying the images and steps for starting the node js server container and Vue cli. The environment for api-server is a connection string using to connect to our MySQL server which is specified in



```
version: '3'
services:
  mysql:
    image: mysql:5.7
    ports:
      - "3306:3306"
    environment:
      - MYSQL_USER=sep6
      - MYSQL_PASSWORD=123456
      - MYSQL_ROOT_PASSWORD=root
      - MYSQL_DATABASE=united_airplanes_db
    volumes:
      - database:/var/lib/mysql
      - ./db/dump:/docker-entrypoint-initdb.d
    networks:
      - sep6_airplanes_network

  api-server:
    build: ./api
    ports:
      - "3257:3257"
    environment:
      - DATABASE_URL=mysql://sep6:123456@mysql:3306/united_airplanes_db
    depends_on:
      - mysql
    networks:
      - sep6_airplanes_network
    volumes:
      - ./api:/usr/src/app

  client:
    build: ./client
    environment:
      - PATH=/usr/local/bin:/usr/bin:/bin:$PATH
    volumes:
      - ./client:/usr/src/app
```

Figure 10 docker-compose.yaml file

depends_on key. The volumes in this case allow us to update the container when saving the code in our IDE.

The reason for using docker-compose for local development was to have a unified environment. That means that none of us had to have MySQL server or Node or anything installed locally on our computer. The only thing was to have Docker and Docker composer installed and the rest was done upon command docker-compose up which built all of our three containers which have all the dependencies of an application that would assist it to run in any environment.

The group's members were each familiar with some type of project board from the internships. The agile development was introduced to us very early in our education, but the full capabilities were never really put into action.

We had a couple of options when choosing, but we ended up trying Jira since none of us knew it before we wanted to try. We used it only to list our issues and then move it around as we were developing the system.

The following screenshots (Figure 11, 12, 13) are showcasing our boards in several stages throughout the project period.

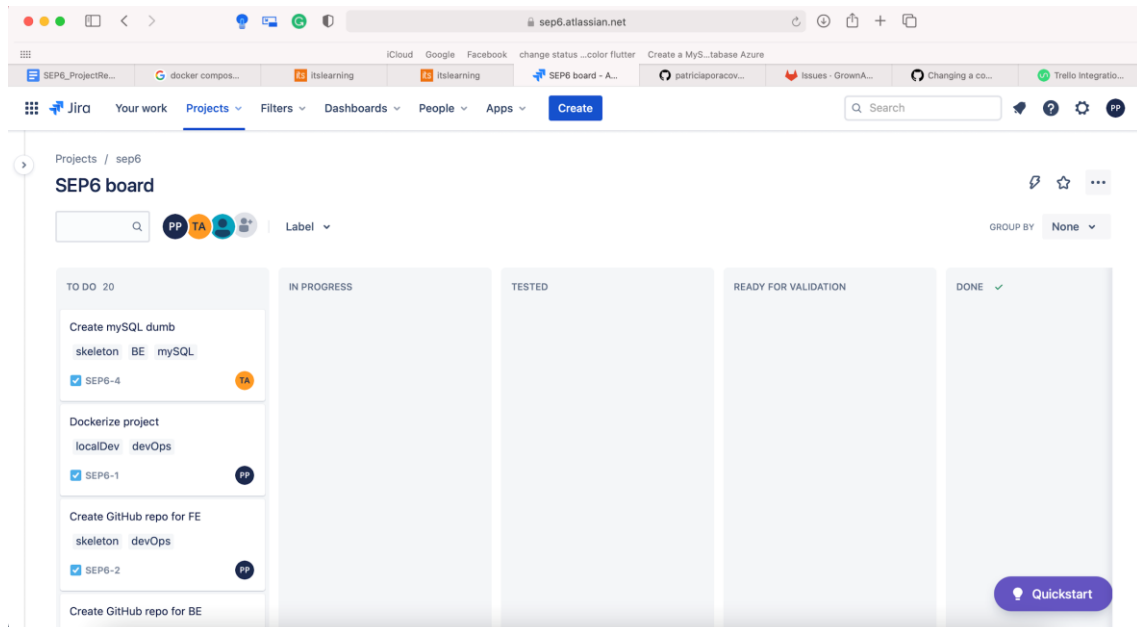
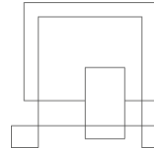


Figure 11 Jira_1

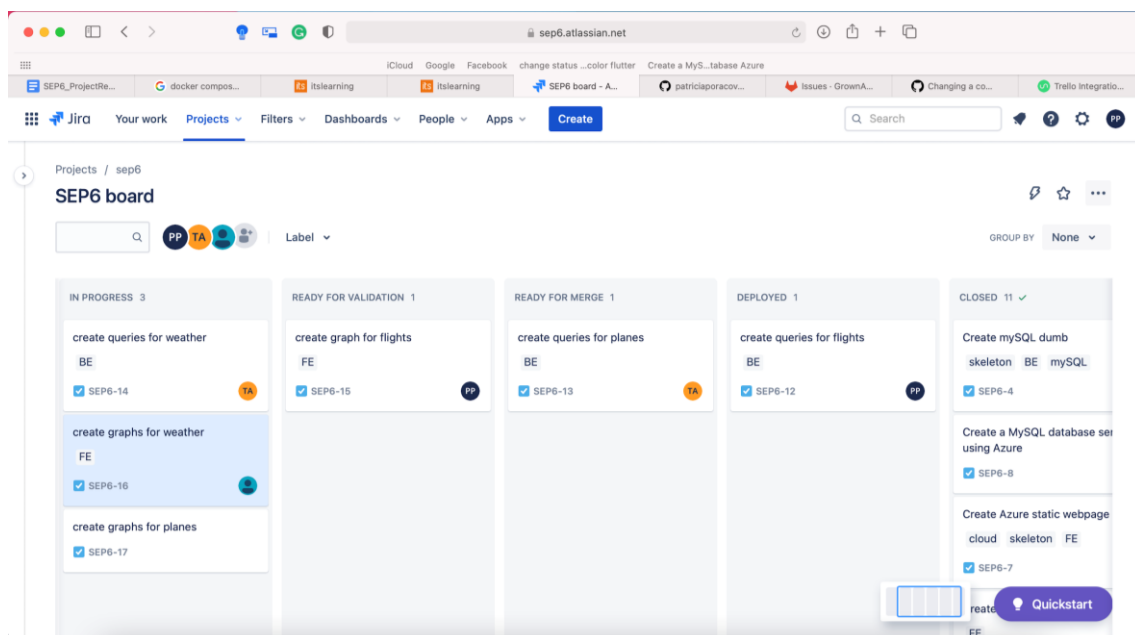


Figure 12 Jira_2

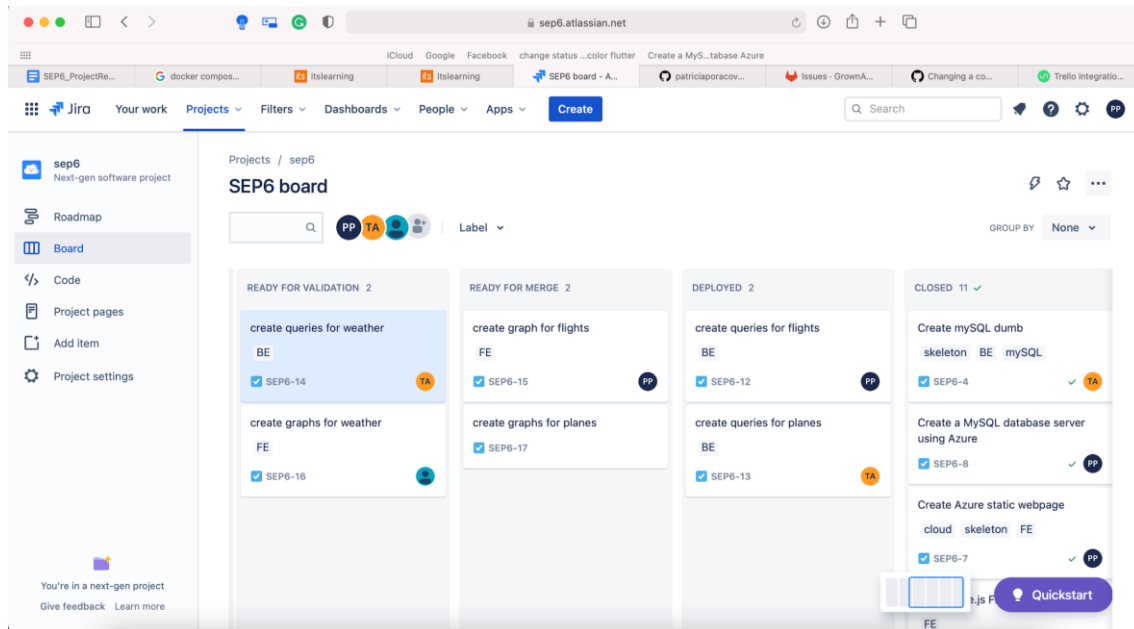
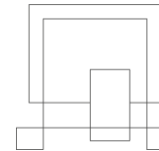
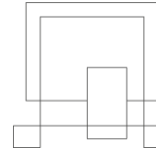


Figure 13 Jira_3

Jira has many options when working with git, so the overview messages or project state or the tasks are very clear. There are always comment sections and labels and basically everything connected to the project. Since our project was small and we were rushing these things because of our problem with google cloud, we did not use it's capabilities fully.

The ideal flow would be to look at the board, take a highest priority issue, move it to the progress column, pull latest code from the master branch, create and checkout a new branch which would start with issue number for easier tracking and some meaningful name. Write our code, push the branch and create pull (or merge, depends on version control) request and assign it to someone. Then move the card to ready for validation and assign someone to look over the code. Then the someone responsible for it will look over the code and either comment or decide the code is ready for merge.

We had this because there are 3 of us and when one pushed both looked over the code just to understand what was happening. This was possible because assignment was small and we were using new technologies and all of us wanted to learn so always all 2 were doing code review and because we had protected the master branch this way we notified the person responsible for master that code is ready. Then the code was merged to master



and developers pulled the latest version of master and continued the work on their branches after rebase.

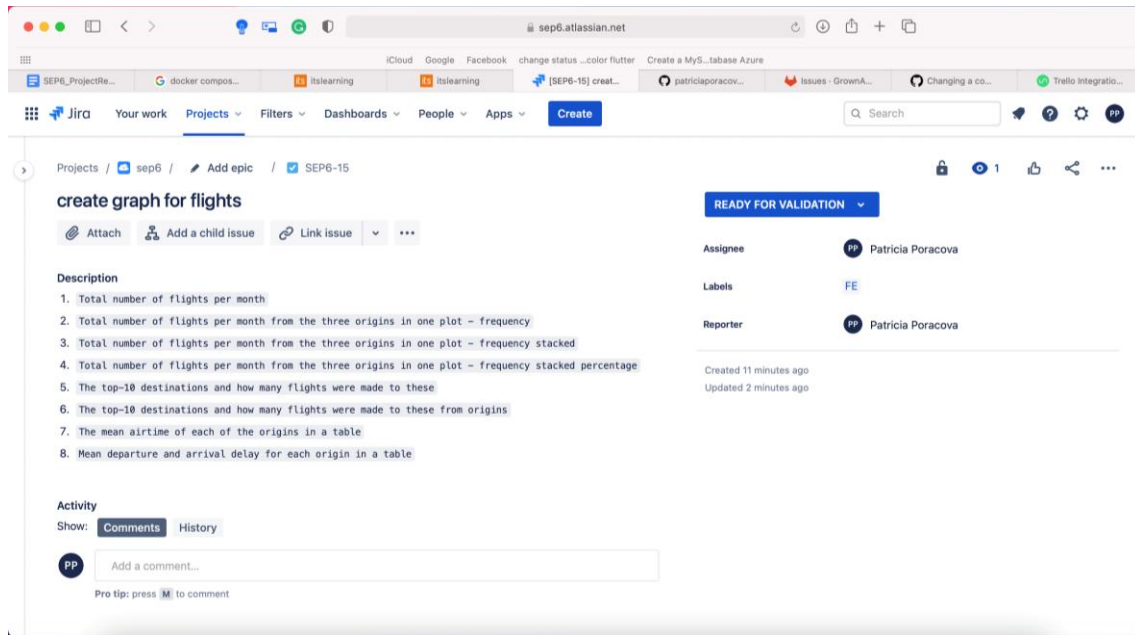


Figure 14 Issue card

This is how the details of the issue card looked like (Figure 14).

Version control

We had two repositories set up on GitHub in which, as mentioned in the first chapter, we had deployment pipelines which were responsible for continuous integration and deployment to our cloud frontend and backend.

Using version control is almost a necessity in development teams and integration of it is implemented to almost every IDE now, also there is GitHub desktop for haters of command line and the portal itself has so many capabilities such as project boards, issue lists, GitHub actions and so on which are making development and deployment automatization so much easier and also helps to create system in individual parts.



To be honest we did not spend much time on testing because of the scope of the project, but ideally, we would want to achieve something like this (Figure 15) using our version control.

3512 handles null description/producer

Status	Pipeline	Triggerer	Commit	Stages	
	#230844914		953147f7 3512 handles null description/producer		00:37:25 15 hours ago
	#230822052		953147f7 3512 handles null description/producer		00:29:50 16 hours ago
	#230821506		18e39eef 3512 handles null description/producer		00:09:21 17 hours ago

Figure 15

In which the automatized pipeline has steps, in this case first it checks code style, then does static analysis of code, then runs tests, then builds, deployed, and runs ui tests. We did not write any tests for a shortage of time but are aware of the process.

The only automation we had with the help of Microsoft azure which was continuous deployment on every push or merge to master. The log below (Figure 16) is from our web service app.

TIME	STATUS	COMMIT ID (AUTHOR)	CHECKIN MESSAGE	LOGS
Tuesday, December 8, 2020				
4:03:46 PM GMT+1	Success (Active)	5363a8a (patriciaporacova)	Merge pull request #1 from patriciaporacova/planes_endpoints planes and weath	
Monday, December 7, 2020				
10:53:46 PM GMT+1	Success	df9889e (patka)	flight routes	
Sunday, December 6, 2020				
1:37:56 AM GMT+1	Success	206d294 (patka)	updating package.json	
1:35:55 AM GMT+1	Success	51d9a9e (patka)	enabling CORS	
1:11:19 AM GMT+1	Success	f8a3625 (patka)	sql config fix	
12:45:27 AM GMT+1	Success	528e3c6 (patka)	sql connection fix	
12:27:25 AM GMT+1	Success	e578910 (patka)	sql connection string updated 3.0	
12:19:19 AM GMT+1	Success	d7a527d (patka)	sql connection string updated 2.0	
12:03:50 AM GMT+1	Success	728ff8b (patka)	sql connection string updated	
Saturday, December 5, 2020				

Figure 16