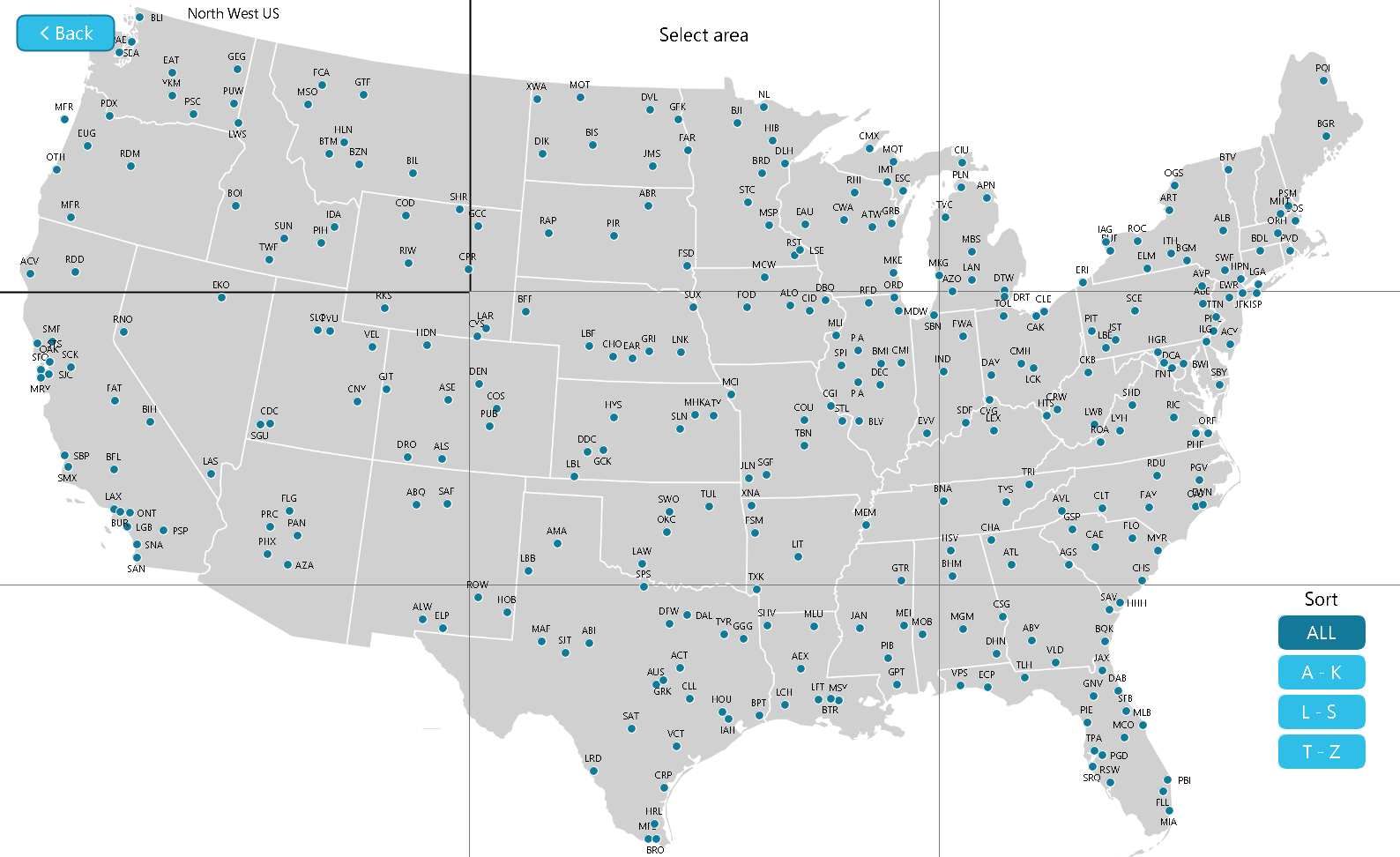
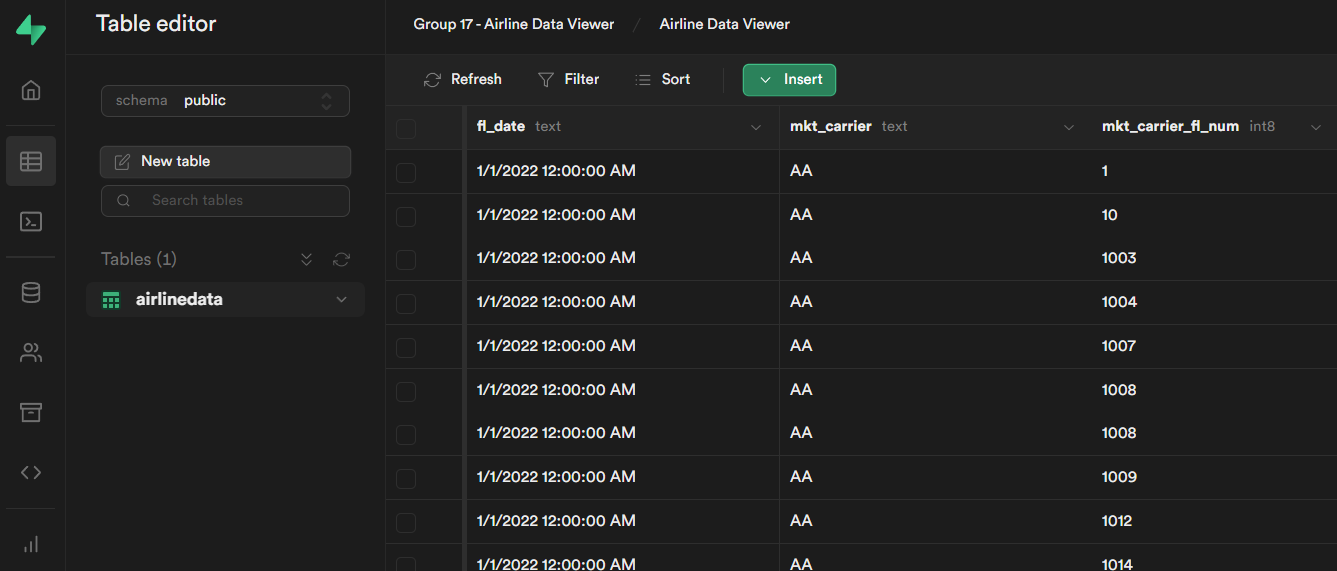
Map Screen

The map screen was the first thing we implemented into the program and was the basis for most of the other features. We began by **mapping all the airports** in the dataset to the map using x and y coordinates. One issue we had was that some airports were too close to each other and appeared as clusters that made specific airports hard to click on. We fixed this issue in three ways: **implementing a zoom screen** that allows for the user to zoom into a section of the map by clicking on it, **creating a filter** that the user can toggle in order to filter the airports using letters of the alphabet and **creating separate screens** for Alaska and Hawaii. The zoom screen was implemented by using a few maths formulas to reposition the airports to the correct area and resizing the map image to fit the required dimensions. Every airport object acts as a widget as it can be clicked and an outline appears when hovered over. To expand the program, we added the **region selection screen** on startup and a **chart selection screen** that appears whenever an airport is selected by the user.

Loading Screen

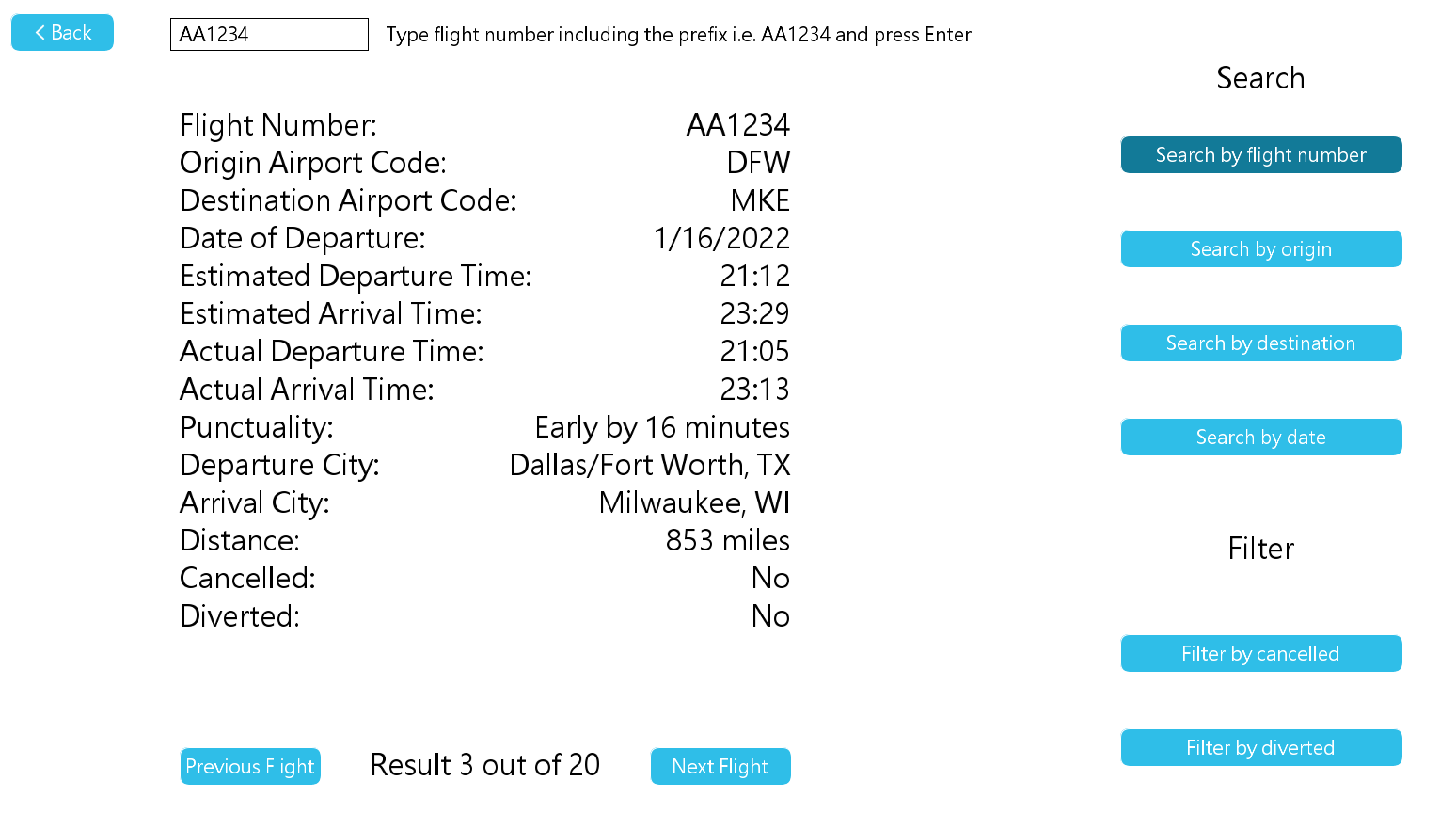
Seeing as our program uses a high amount of data, the startup takes a while. Instead of leaving the user waiting on an empty screen, we decided to **implement a loading screen**  to show that there is progress in opening the application. This was done using a new thread in order to run both the animation and data loading at the same time. Seeing as processing does not support .gif files, it was necessary to split the animation up into separate frames that the draw function iterates through.

SQL Implementation

Our first idea for sorting through the data provided was simply reading the file and dividing the data into different variables in a flight object that could be accessed throughout the program. Although this idea worked, we believed we could create a smoother, faster experience for the user (especially when working with larger datasets, **it allowed us to work with the biggest data file available to us for the project**) by **implementing SQL** into our application. We began by **setting up a local server using PostgreSQL** and the BezierSQLib in order to learn how to use the language and trying out a few queries. The next step was completely **replacing the code** that used the old flight object with new SQL queries to connect both programs. In retrospect, it would have been better if we did this earlier on in the development as it would have saved us a lot of time. The final step was **changing the server** from a locally hosted one, to one that could be accessed by any user on any device. Thanksto **SupaBase**’s open source Postgres database we could implement this idea into our program.

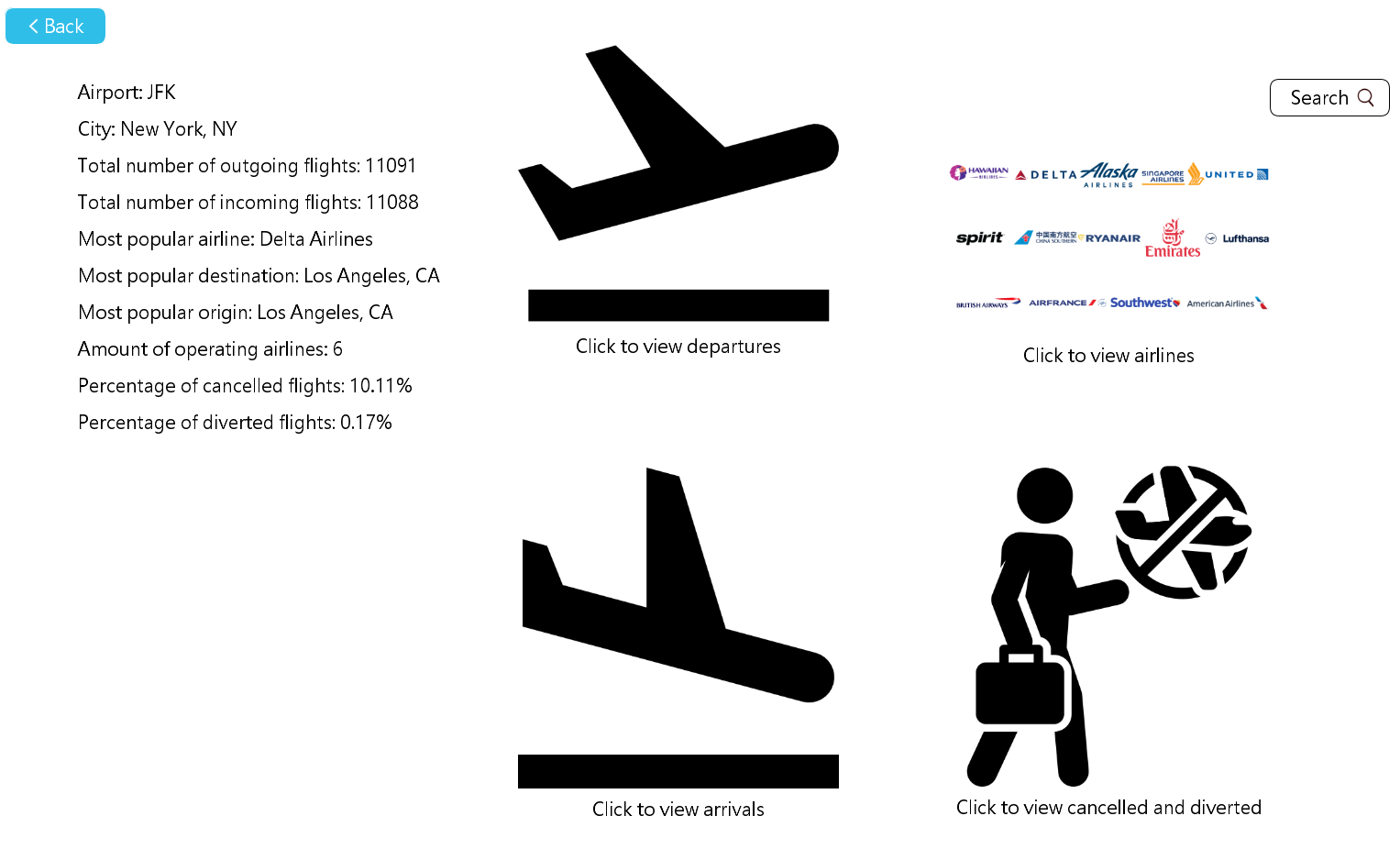
Search Bar Function

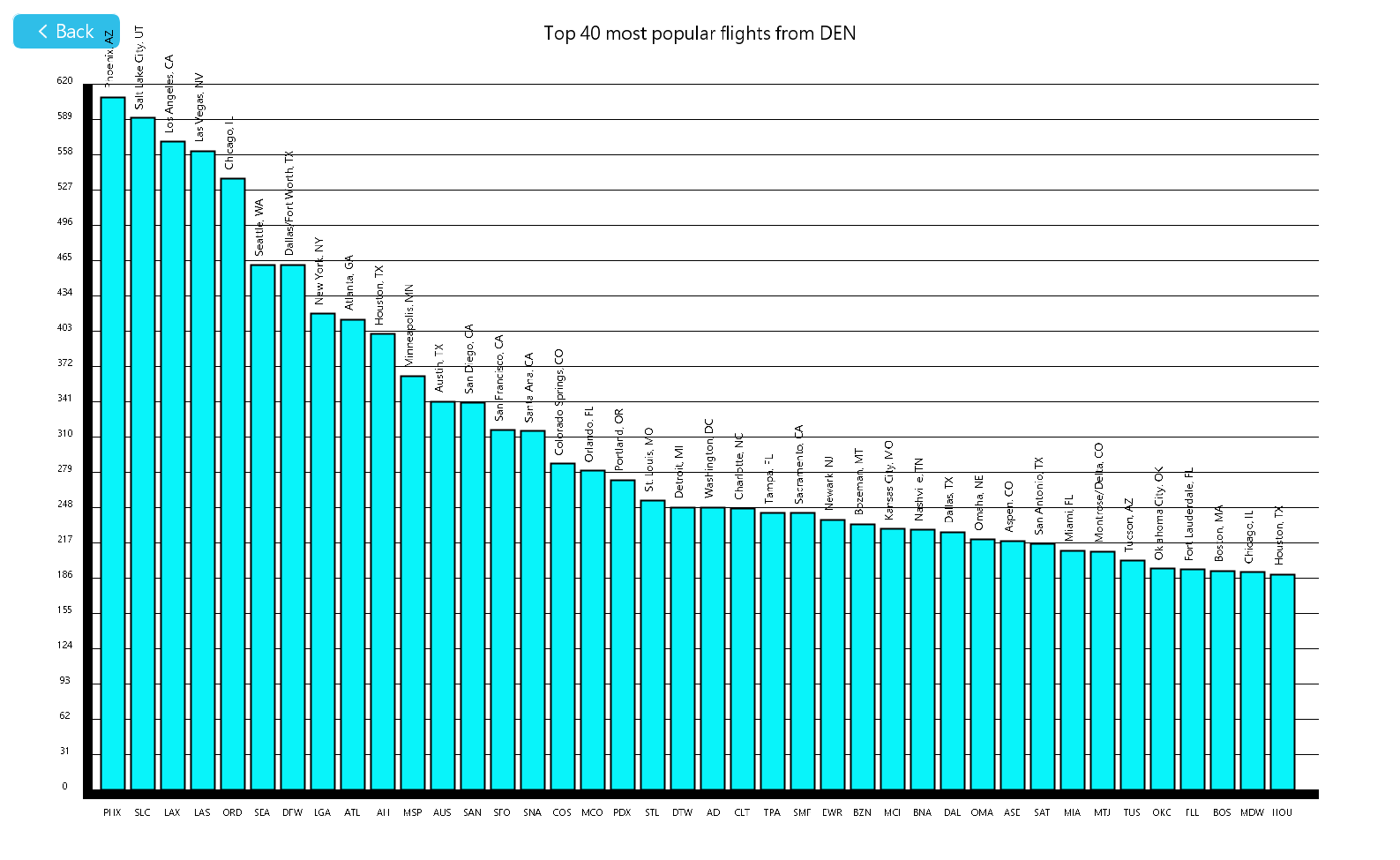
As well as our map, we wanted to implement a way to access data withmore accuracy. To do this we decided to **create a search bar** function, which coincides with other customizable options and filters, such as a way to search by Flight Number, Airport Origin, Date or Destination, and functions to further condense these results with filters of “**cancelled**” and “**diverted**”. These functions work by returning a specific “case” when a widget is selected, which in turn runs a specific interface to match the user’s selection. With one selected, a user can search for a specific flight by pressing “enter” or selecting a date from the dropdown menu. The data returned is **any** dataset that matches the user’s input. The search bar works by adding whatever keys are pressed by the user to a string, as long as they are letters. Letters are subtracted by pressing the “backspace” button. These keys are detected by the keyPressed() function in the main which calls the searchTyping() function which includes if statements the decides what will happen when a certain key is pressed.

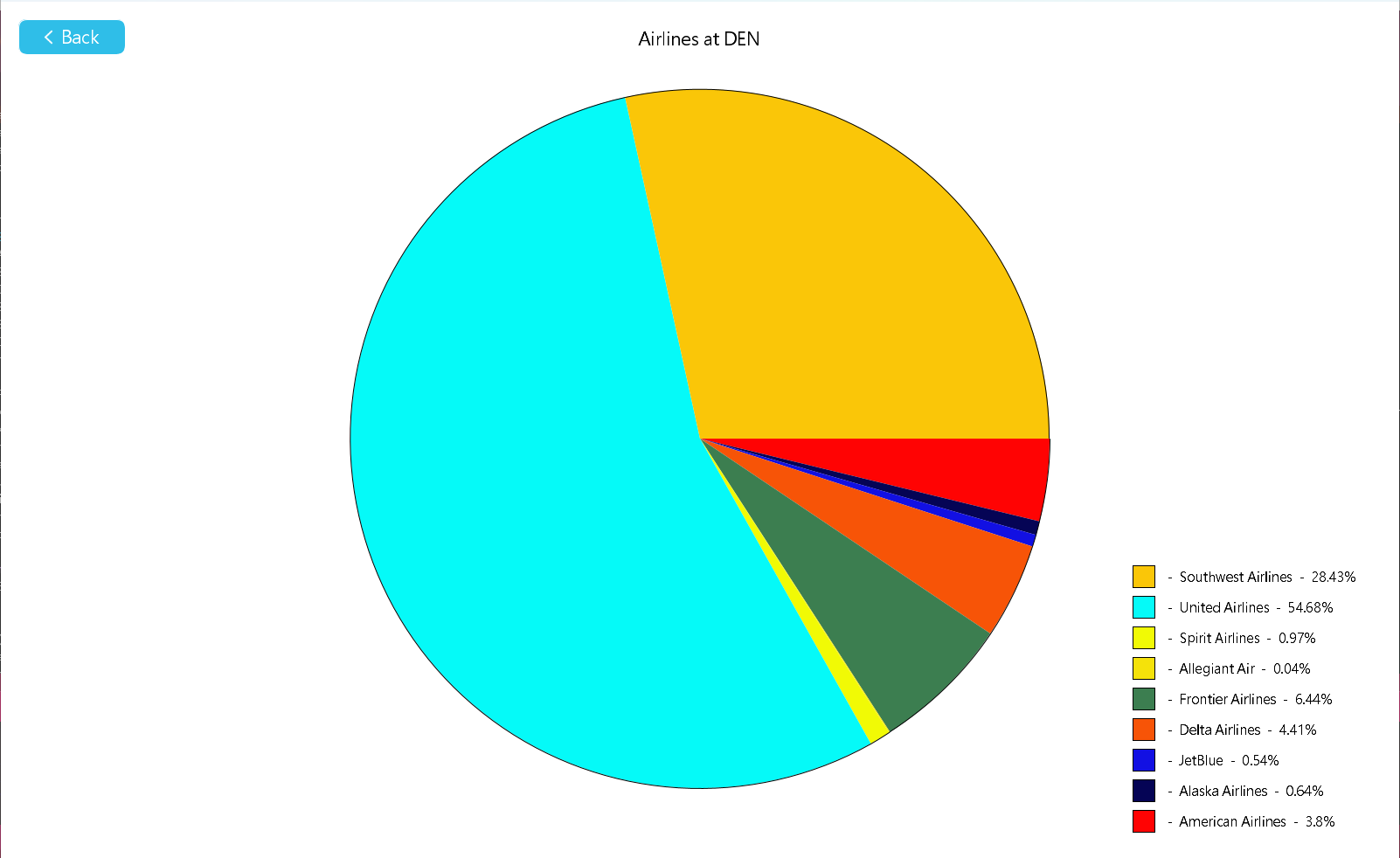
To ensure a proficient user experience and increased flexibility when viewing large amounts of flight data, we have implemented a scroll function within our program. This feature allows users to easily navigate through extensive search results and find the specific flight they are looking for. For instance, if a search for flight number "AA1234" returns 20 results, the user can simply scroll through the list or use the "Next flight" or "Previous flight" buttons to navigate through the results. To provide users with the most relevant information, each index of the list displays a set of useful data, including estimated and actual arrival times, punctuality, and other pertinent details. By presenting this information in a clear and organised manner, users have greater control over the data they view and can quickly access the specific material they need from the program. Our scroll function not only streamlines the search process, but also improves the overall user experience by allowing users to easily and efficiently navigate through extensive amounts of flight data.

Our program offers an easy-to-use and user-friendly search functionality that is accessible from both the main screen and within each airport accessed. By providing this feature in both locations, users can perform quick and effortless searches from anywhere within the program where data is accessed, without having to navigate through multiple menus or screens. When the search bar is accessed within a specific airport, users are fluidly taken to a dedicated search screen that displays the vast data set of flights available for that airport. This not only provides users with comprehensive information, but also saves them time and effort. We have carefully integrated the search bar with the rest of the program to make sure that users have a smooth and coherent experience while using our program. We have created a program that both meets and exceeds the needs of users who may require access to a specific piece of information in vast amounts of flight data.

Charts and Graphs

To add another layer to our project we wanted to further our analysis of the data and dive deeper again.To do this we decided to create **bar charts** and **pie charts**, this allowed us to **visualise** the data for the user and create a simple and effective way to view the given data for each of the corresponding airports. We implemented all of these features onto our **chart selection screen**. Once the user is on the map and selects an airport to view they are brought to the chart selection screen. This screen allows us to give the user a wide variety of choice of visual and small data, it also allows the user to select the **search bar** icon on the top right which will redirect the user to the search bar screen and automatically input the **origin** of the airport into the search bar for the user and display the corresponding data for that airports origin on the search bar screen. As you can see when the user is brought to this screen they are presented with **four different choices of graph** and also a piece of small data on the side. This small data gives a **summary** of the main statistics corresponding to the airport that was selected, this is then visualised in more depth in each of our four graphs that we have created. To create a **realistic** visualisation of the data we picked the main statistics that users would want to access/research and converted them into graphs.



The first graph we created displays all of the **departing flights** from the given airport to the other airports included and their corresponding cities also on top of each bar of the chart. We displayed the 40 most popular airports at most for one airport in order to keep the bar chart tidy and visually pleasing for the user. We also implemented a bar chart to visualise the **arriving flights** to the given airport that is selected which shows the flights from other airports to the given airport and their corresponding cities. To further our analysis again we decided to implement pie charts to envision two more sets of data for the selected airport that the user has chosen. The first of the two being the most popular **airlines** that fly to/from the selected airport. The pie chart allows us to easily decipher which airlines are popular in which airports and which ones aren’t. As it can be seen there is also a **key** down on the bottom right hand side of the pie chart screen which shows the airlines and their corresponding percentages too. Lastly we also implemented a pie chart which displays the **cancelled, diverted and unchanged flights** for the given airport. This pie chart also has a key, the unchanged section of the pie chart usually should be 90% or above in order for the airport to be reliable and trustworthy for booking flights. Each of the pie charts and bar charts were implemented using our SQL table to extract the correct data for each airport and are drawn using a series of functions. 

TO BE NOTED:

When we were pushing/uploading our code onto GitHub, mainly Radek or Reece pushed it onto the main GitHub branch. We did it in a way that it was Radek’s or Reece's job to merge the new code with the old code or whatever else was needed to merge the code etc. outside of GitHub/Git and then test the code before pushing the code onto the main GitHub branch, we found this most efficient in preventing merging errors and to keep GitHub tidy for us, keeping it that the majority of the work was pushed by only two people onto the GitHub main branch. This obviously led to mainly Radek and then Reece actually uploading the group's code to the main GitHub branch but in fact everyone in the group did the same amount of work. For example, for a certain feature we had to implement like the chart screen up to 3 people would work on the chart screen code and then e.g. Radek would be given the code and he would merge that code with the code we already have and check that it’s working correctly with our code that we already have, then it would be pushed on if it worked correctly. Overall, we did more or less the same amount of work for this group project. We thought that it would be best if we clarified it in our report to prevent confusion when the person who is marking our project looks at our GitHub repository.