Use Cases and Queries:

1. View Public Info
    - ***Search & Flight status***

      Frontend: On the landing page of the site, we utilized the list component to render the list of future flights with details and purchase buttons on the sides. For users that are not logged in, the purchase button is disabled and he/she can only see the details of the flight (without the purchase button in the detail page). Every time the home page is opened/accessed, the frontend makes a get request to the backend so that we can pull the future flight data from the backend.
      To view the flight status, the user can either view the abstract list view or by clicking into the flight detail to view more information.

      Backend: Make a get request (route: futureFlights) to the database server. Everytime the route is accessed, it sends out a query to the database server and retrieves the data and stores it into a temporary variable named rows, and sends it back to the frontend. The error handling was accomplished by adding an else statement in the callback function.

2. Register
    - ***Customer sign up***

      Frontend: We have a form field that requires the user to enter all required information in order to sign up. Validation is performed throughout the process, and if any of the fields were left out, the input box would be highlighted and if the user clicks the sign up button without entering all the fields, an alert would pop up to prevent the user from signing up. This is accomplished by implementing a set of rules for all the fields and dynamically checking them on the fly.

      Backend: Make a post request (route: newCustomer) to the database server, and pull out all the input information from the body. Parse the dob and passport expiration date into standard SQL format and hash the password. Then write a SQL query to insert all the values into the database if the customer doesn't exist.

    - ***Staff sign up***

      Frontend: Similar logic to the customer sign up.

      Backend: Make a post request (route: newStaff) to the database server, and pull out all the input information from the body. Parse the dob into standard SQL format and hash the password. And then write a SQL query to insert all the values into the database if the staff doesn't exist.

3. Login
    - ***Customer login***

Frontend: Present two input fields for users to enter their email and password. The input information was then sent to the backend to perform the check there.

Backend: Make a post request (route: loginCustomer) to the database server, and pull out all the input information from the body. Hash the password into the correct format and check against the saved password from the database. If the customer doesn't exist or the password doesn't match the one stored, there will be a corresponding message sent back to the frontend and displayed there.

- ***Staff login***
  Frontend: SImilar logic as customer login

  Backend: Make a post request (routes: loginStaff) to the database server, and pull out all the input information from the body. Hash the password into the correct format and check against the saved password from the database. Then send a SQL query to get the values from the database. If the staff doesn't exist or the password is doesn't match the one stored, there will be a corresponding message.

- ***Authorization***
  Frontend: Every time the page is accessed, the site first checks the local storage and sees if the machine has a local token stored already. If there indeed exists a token, then the token is sent back to the backend and compared there. Once the site got the result from the backend, it would either log the user in or stay in not logged in status. This happens before the entire page is rendered.

  Backend: Make a get request (route: auth) to the database, and check if the user is authorized. jwt token is used. It sends back to the front end as a token with an expiration period of one hour, every time the user accesses the page, it will automatically make a network request to the back end with the token that is being pulled from local storage (this is in the frontend), and check against the token that we stored in the back end. If the token id matches, we authorize the user to log in automatically.

- ***Logout***
  Frontend: There is a logout button on the top navbar if the user is logged in. Every time the logout button is clicked, there will be a "Bye" alert message to inform the user that they logged out.

  Backend: Make a post request (route: logout), and delete the token from local storage.

4. Customer
   - ***View My Flights***

Frontend: Under the customer profile page (after a customer is logged in), the default view would be all the past flights he/she has taken. He/she can also click on the future flight tab to view the future flights.

Backend: Make a post request (route: customerFlights) to the database server. send a SQL query to get the desired value from the database and send the results back to the frontend.

- ***Search for Flights***
  The same search page was used as the public view info section.

- ***Purchase Tickets***
  Frontend: The user can either go the home page and view all the displayed future flights and click the purchase button on the right end, or can search for specific flights, and click the purchase button to go to the purchase page and fill out the form accordingly. When the user successfully purchased the ticket, there will be an alert message saying “Purchase success”, and direct back to the home page.

  Backend: Make a post request (route: flightPurchase) to the database server. First, insert a ticket instance to the ticket table, and if there’s no error in inserting the data, add a purchase record into the purchase table.

- ***Review***
  Frontend: There will be a review button shown next to all the flight records list. Once it’s clicked, it will direct to the review page, and allow the user to enter ratings and comments. If the flight is in the future, there will be an alert message saying “you can not review future flight” when the user clicks the submit button. Whatever the response got back, whether it’s “success” or “already rated”, it will be displayed as an alert message.

  Backend: Make a post request (route: addReview) to the database server. Try to insert it into the feedback table. If there’s a duplicate entry, send back a 500 response and error message saying “already rated”.

- ***Track Spending***
  Frontend: In terms of UI, the user has 3 tabs to click under the spending tab in the profile page. They are Monthly View/ Last Year/ Specify Dates. Under the Monthly View, it shows monthwise spending as a table format. Under the Last Year tab, it’s going to be a number with a dollar sign. Under specify Dates, it asks the user to input a range of dates, and it will render the result in both monthwise table and a total spending number. The way this is accomplished is by first getting a list of all the tickets the user purchased last year, and it will calculate the spending according to the list of tickets. If the user wants to view the spending for

the specified range of dates, it will make another network request to get all the tickets and calculate the monthwise spending similar to the Monthly view tab.

Backend: Make a request (route: custTickets) to the database server with the specified range of dates in the request body. The default starting date will be one year from now. The default end date is today.

5. Staff
   - ***View Flights***
     Frontend: The staff can view flights in 2 different tabs. One is to search for flights to view all the flights (past/future/current) within the airline he/she works for. Under the staff's profile page, he/she will be able to view all the flights in the next 30 days within the airline he/she works for. In the search page, search by a range of dates is restricted so that only staff members can get access and search under this tab.

     Backend: Make a get request (route: futureFlights), and write a query to get all the future flights information based on the input, and send the results back. If he/she is trying to view the past flights or view flights with more attributes, the route used is "searchByDateRange".

   - ***Create New Flights***
     Frontend: In the staff profile page, under the Add New tab, and in the flight tab, there's a form to fill out all the required information for adding a new flight. There is a set of rules to validate the form fields. After clicking submit button, there will be an alert message for either success or failure.

     Backend: Make a post request (route: addFlight) to the database server. It will first check if all the fields are valid (such as airport, airline, etc.). If all the checks pass, it will create a flight in the flight table. And send the response back to the frontend.

   - ***Change Status***
     Frontend: In the staff profile page, under the All Flights tab. There is an "Edit status" button that enables staff to click on it. Every time the staff clicked, there will be an alert message telling you the next steps to view the status.

     Backend: Make a post request (route: toggleFlightStatus) to the database server. Find the flight information first, and make a SQL query to update the status.

   - ***Add New Airplane***
     Frontend: In the staff profile page, under the Add New tab, and in the airplane tab, there's a form to fill out all the required information for adding a new airplane. There is a set of rules to validate the form fields. After clicking submit button,

there will be an alert message for either success or failure, and it will show under the form.

Backend: Make a post request (route: addAirplane) to the database server. It will first check if all the fields are valide (such as airport, airline, etc.). If all the checks pass, it will create a new airplane on the airplane table. And send the response back to the frontend.

- ***Add New Airport***
  Similar logic to Add New Airplane.

- ***View Feedback***
  Frontend: Under the “Search for Flight” button, the staff can search for a specific flight and click on the “View Ratings” button. It will direct to the rating page which shows all the customers who have taken this flight, and their ratings, comments, and average rating for this flight.

  Backend: Make a post request (routs: getAllRatings) to the database server. Send a query to get all the feedbacks from the feedback table based on the specified flight information.

- ***View Reports***
  Most Frequent User
  Frontend: On the report page, there is a Most Frequent Customer tab on the left side of the page. It shows the most frequent customer information at the top, and all the flights this customer has taken.

  Backend: Make a post request (route: mostFrequentCustomer) to get the most frequent customer by counting the number of purchases that has the airline name that the staff works for (Acoomplished by writing subqueries), and send the customer information to the frontend. After getting the customer, another network request will be received from the frontend to get all the flights taken by this customer.

  Revenue Earned
  Frontend: On the report page, there is a Total Revenue tab on the left side of the page. It shows the total revenue Last Year and Last Month at the top. At this moment, we have a list of all the tickets in our central state management file. The revenue is calculated by summing up all the sold prices within the corresponding dates

  Backend: Make a post request (route: airlineTickets) to the database server. It handles the revenue by querying a list of all the tickets sold last year under the

specific airline, and the result is sent back to the frontend where more complicated calculations are performed.

Tickets Sold

Frontend: On the report page there's a tab for tickets sold. Three different views are available for different needs, which are the last 6 months (table format), last year (number), and specified range of dates (number). They are dynamically rendered by accessing all the tickets sold last year and rendering them accordingly. For the specified date, it will make a separate request to the backend and retrieve the count of all the tickets within that range

Backend: Make a request (route: airlineTickets) to the database server to query for all the tickets sold last year and send it directly back to the frontend where more complicated parsing is performed. For tickets sold in a range of dates, it sends out a more complicated query to pull a list of tickets from the database and count the number of instances and send that data back to the frontend.

Flights Delayed

Frontend: Flights delayed would be a listview under two tabs, where one is for the last 6 months and one is for last year. Every time the report page is accessed, the frontend sends out a network request to prepare all the flight data and render only the delayed flights within the specified range of dates when the tab is selected.

Backend: Make a get request (route: allFlights). It gets all the flight data by querying the flight table for that particular airline and sends it back to the frontend for filtering. Since the flight data was very versatile in many use cases, we did not write a separate query for just grabbing the delayed flights. Storing all the flight data in the state management prevents multiple network requests and grants better performances.

Top Destinations

Frontend: Two tabs under the top cities tab in the report page render 2 tables that contain the ranking of each city by how frequent/popular they are.

Backend: Make a post request (route: topThreeArrivalAirports) to the database server, and send a query to count the frequency of all the destinations of sold tickets in a specified range, and then send the results back.