Here are how the important keys are identified (with data examples):

<u>Collection: airlines</u>

```
name:           "Southwest Airlines"
```

*All flights will have an **airline** from the airlines collection like above.*

<u>Collection: airports</u>

```
"iataCode":                 "ORD",
"name":                     "Chicago O'Hare International Airport",
"city":                     "Chicago",
"country":                  "United States",
"location":     "coordinates": {
                      0:         -87.9048,
                      1:          41.9786
"timezone":                 "America/Chicago"
```

*All flights will have an **arrivalAirport.name, arrivalAirport.iataCode, departureAirport.name, departureAirport.iataCode** from the airports collection like above.*

<u>Collection: flights</u>

```
"arrivalAirport": {
                "name":         "Salt Lake City International Airport",
                "iataCode":     "SLC"

"dayOfWeek":                    "Monday"

"departureAirport": {
                "name":         "Spokane International Airport",
                "iataCode":     "GEG"

"departureTime":                "2025-07-14T02:40:00-07:00"

"airline":                      "Delta Air Lines"

"arrivalTime":                  "2025-07-14T06:27:00-06:00"

"flightNumber":                 "3787"
```

Query examples:

```
/*** Collection: airlines ***/

// Find all airlines whose name includes a user-provided keyword
db.collection("airlines")
  .find({ name: new RegExp(userSearchTerm, "i") })
  .toArray();


/*** Collection: airports ***/

// Find airport by selected IATA code
db.collection("airports").findOne({ iataCode: selectedIataCode });

// Find airports by selected name
db.collection("airports").find({ name: selectedAirportName }).toArray();

// Find airports in a selected city
db.collection("airports").find({ city: selectedCity }).toArray();

// Find coordinates for a selected airport
db.collection("airports").findOne(
  { iataCode: selectedIataCode },
  { projection: { "location.coordinates": 1 } }
);


/*** Collection: flights ***/

// Convert userStartDate to day of the week (used in many queries)
const userDayOfWeek = new Date(userStartDate).toLocaleDateString("en-US", {
  weekday: "long",
});

// Find all flights departing from a selected airport code
db.collection("flights")
  .find({ "departureAirport.iataCode": selectedDepartureIata })
  .toArray();

// Find flights from a selected airport operating on the user's selected day
db.collection("flights")
  .find({
    "departureAirport.iataCode": selectedAirportCode,
    dayOfWeek: userDayOfWeek,
  })
  .toArray();
```

```
// Find a flight by its number at a specific departure airport
db.collection("flights")
  .find({
    flightNumber: selectedFlightNumber,
    "departureAirport.iataCode": selectedIataCode,
  })
  .toArray();

// Find all flights at a selected airport on a selected day for a selected airline
db.collection("flights")
  .find({
    "departureAirport.iataCode": selectedAirportCode,
    dayOfWeek: selectedDay,
    airline: selectedAirline,
  })
  .toArray();
```

<u>Useful routes</u>:

Retrieve all airlines:

```
// routes/airlines.js

const express = require("express");
const router = express.Router();

module.exports = (db) => {
  router.get("/", async (req, res) => {
    try {
      const airlines = await db
        .collection("airlines")
        .find({})
        .sort({ name: 1 }) // sort alphabetically
        .toArray();
      res.json(airlines);
    } catch (err) {
      console.error("Error fetching airlines:", err);
      res.status(500).json({ error: "Internal Server Error" });
    }
  });

  return router;
};
```

Retrieve flights (departure airport, arrival airport, departure date begin, departure date end)

*This will compute the **dayOfWeek** from **departureTime** Begin and End range dates, and return all flights from the selected **departureAirport** going to the **arrivalAirport** on those days.*

```
// routes/flights.js

const express = require("express");
const router = express.Router();
const { DateTime } = require("luxon"); // For date handling

module.exports = (db) => {
  router.get("/search", async (req, res) => {
    const { departureAirport, arrivalAirport, departureBegin, departureEnd } =
      req.query;

    if (
      !departureAirport ||
      !arrivalAirport ||
      !departureBegin ||
      !departureEnd
    ) {
      return res
        .status(400)
        .json({ error: "Missing required query parameters" });
    }

    try {
      const begin = DateTime.fromISO(departureBegin);
      const end = DateTime.fromISO(departureEnd);

      if (!begin.isValid || !end.isValid || begin > end) {
        return res.status(400).json({ error: "Invalid date range" });
      }

      // Get all days of the week in range
      const dayOfWeekSet = new Set();
      let current = begin;
      while (current <= end) {
        dayOfWeekSet.add(current.toFormat("cccc")); // e.g. "Monday"
        current = current.plus({ days: 1 });
      }

      const daysArray = Array.from(dayOfWeekSet);

      const flights = await db
        .collection("flights")
        .find({
          "departureAirport.iataCode": departureAirport,
          "arrivalAirport.iataCode": arrivalAirport,
          dayOfWeek: { $in: daysArray },
        })
        .toArray();

      res.json(flights);
    } catch (err) {
      console.error("Error searching flights:", err);
      res.status(500).json({ error: "Internal Server Error" });
    }
  });

  return router;
};
```

Build flight data for chosen flight

This will provide the flight departure and arrival times in local time and the departure and arrival date(s) for the flight as requested (not the past date stored in the database). Flight duration is also computed, accounting for timezone differences.

```
// routes/flights.js

const express = require("express");
const router = express.Router();
const { DateTime } = require("luxon");

/**
 * POST /api/flights/compute
 * Body: {
 *   flight: { departureTime: ISOString, arrivalTime: ISOString },
 *   date: "YYYY-MM-DD"
 * }
 */
router.post("/compute", (req, res) => {
  const { flight, date } = req.body;

  if (!flight || !flight.departureTime || !flight.arrivalTime || !date) {
    return res.status(400).json({ error: "Missing flight data or date" });
  }

  try {
    const dtDepOriginal = DateTime.fromISO(flight.departureTime);
    const dtArrOriginal = DateTime.fromISO(flight.arrivalTime);
```

```javascript
    const depTimePart = {
      hour: dtDepOriginal.hour,
      minute: dtDepOriginal.minute,
    };
    const arrTimePart = {
      hour: dtArrOriginal.hour,
      minute: dtArrOriginal.minute,
    };

    const dtDepAdjusted = DateTime.fromISO(date, {
      zone: dtDepOriginal.zone,
    }).set(depTimePart);
    let dtArrAdjusted = DateTime.fromISO(date, {
      zone: dtArrOriginal.zone,
    }).set(arrTimePart);

    if (dtArrAdjusted < dtDepAdjusted) {
      dtArrAdjusted = dtArrAdjusted.plus({ days: 1 });
    }

    const duration = dtArrAdjusted
      .diff(dtDepAdjusted, ["hours", "minutes"])
      .toObject();

    res.json({
      departureTimeLocal: dtDepAdjusted.toFormat("yyyy-LL-dd HH:mm ZZZZ"),
      arrivalTimeLocal: dtArrAdjusted.toFormat("yyyy-LL-dd HH:mm ZZZZ"),
      durationFormatted: `${Math.floor(duration.hours)}h ${Math.round(
        duration.minutes
      )}m`,
      durationMinutes: Math.round(
        dtArrAdjusted.diff(dtDepAdjusted, "minutes").minutes
      ),
    });
  } catch (err) {
    res.status(500).json({ error: err.message });
  }
});

module.exports = router;
```