CCT College Dublin Continuous Assessment

Group Report

Module Title:	Data Storage Solutions Commercial Solutions Design		
Assessment Type:	Group (3)		
Assessment Title:	CA: 1		
Lecturer Name:	Muhammad Iqbal David McQuaid		
Students Names:	Yuri Ribeiro	Xiaohui Weng	Leisly Pino
Students Numbers:	2020347	2020387	2020303
Assessment Due Date:	07 / May / 2023		
Date of Submission:	12 / May / 2023		
GitHub Depository	https://github.com/lepidu/CSD-DSS		
Google Docs	CSD&DSS.docx		

Declaration

By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution.

Index

Dataset

As we know that Hadoop works with big data, and also based on the CA requirements we need to use a dataset which should not be older than 5 years. The dataset that was provided 2 years ago is about US air carriers' information, which includes date, time, origin, destination, airline, distance, and delay status of flights from 2016 to 2018. It is estimated that air transportation delays put a 4 billion dollar dent in the country's gross domestic product that year.

We opted to not use all the columns, so the data that will be displayed on our application is:

1. **ID:** BSon ObjectId
2. **Year:** 2016, 2017, 2018
3. **Month**: 1-12
4. **DayofMonth:** 1-31
5. **DepTime**: Actual departure time (local, hhmm)
6. **CRSDepTime:** Scheduled departure time (local, hhmm)
7. **ArrTime:** Actual arrival time (local, hhmm)
8. **CRSArrTime:** Scheduled arrival time (local, hhmm)
9. **Origin:** Origin IATA airport code
10. **Dest:** Destination IATA airport code
11. **Distance:** In miles

(For more information click here: Dataset link)

Figure 1. Variables in the Domestic Flight Database.

Data Storage framework

Data Storage Framework is a software which is responsible for computing the data in the system, in another word, it manages the data in a way to process or store the data. Hadoop is one of the most famous open source software for the big data. It provides some services to solve the big data problems, such as massive data storage by using HDFS, resource management, scheduling and allocation by using YARN etc..As far as our knowledge is concerned, that there are 3 important serivers for HDFS, the purpose of that to provides storage for the massive amount of data.

a. NameNode(master node) that maintain the whole file system
b. Secondary nameNode is a file system Image, when NameNode privodes services for the cluster and sometimes may not provide sufficient resources, then Secondary nameNode will take this part of the job.

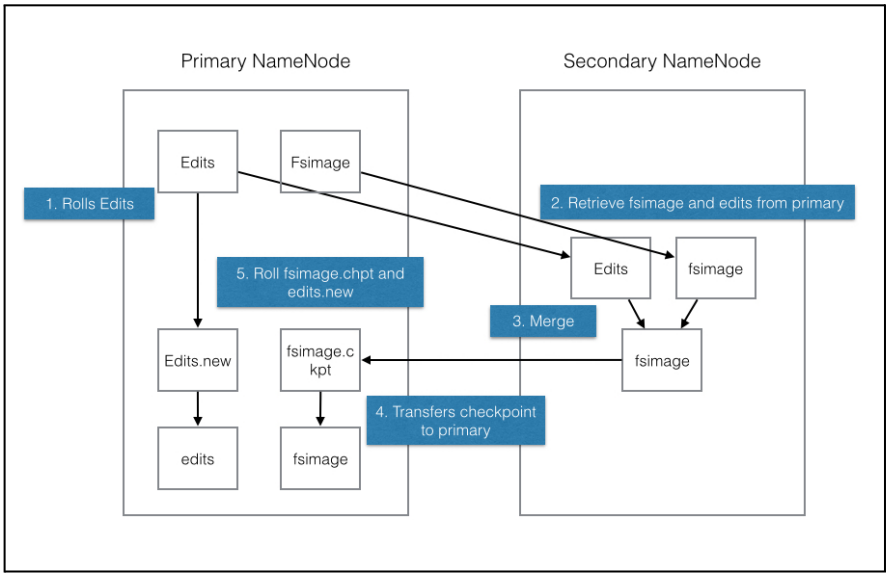

Figure 2. Primary and secondary node diagram.

c. DataNode (Slave node) that when the client requests to write a file block, those blocks will be copied into multiple dataNodes, in general, the size of each block will be 128MB. And how many copies will be executed will depend on the client. As you can see figures down below that hadoop is running properly.

```
hduser@xiaohui-VirtualBox:~/Desktop/Hadoop$ start-dfs.sh
Starting namenodes on [localhost]
Starting datanodes
Starting secondary namenodes [xiaohui-VirtualBox]
hduser@xiaohui-VirtualBox:~/Desktop/Hadoop$ start-yarn.sh
Starting resourcemanager
Starting nodemanagers
hduser@xiaohui-VirtualBox:~/Desktop/Hadoop$
```

Figure 3. Initiation of Hadoop.

On the other hand, MapReducer provides the computing Framework to deal with the data. When MapReduce receives a task, firstly the input from HDFS will split the task into Map task in different nodes, then it will shuffle all those value and reducer them, then get the output.

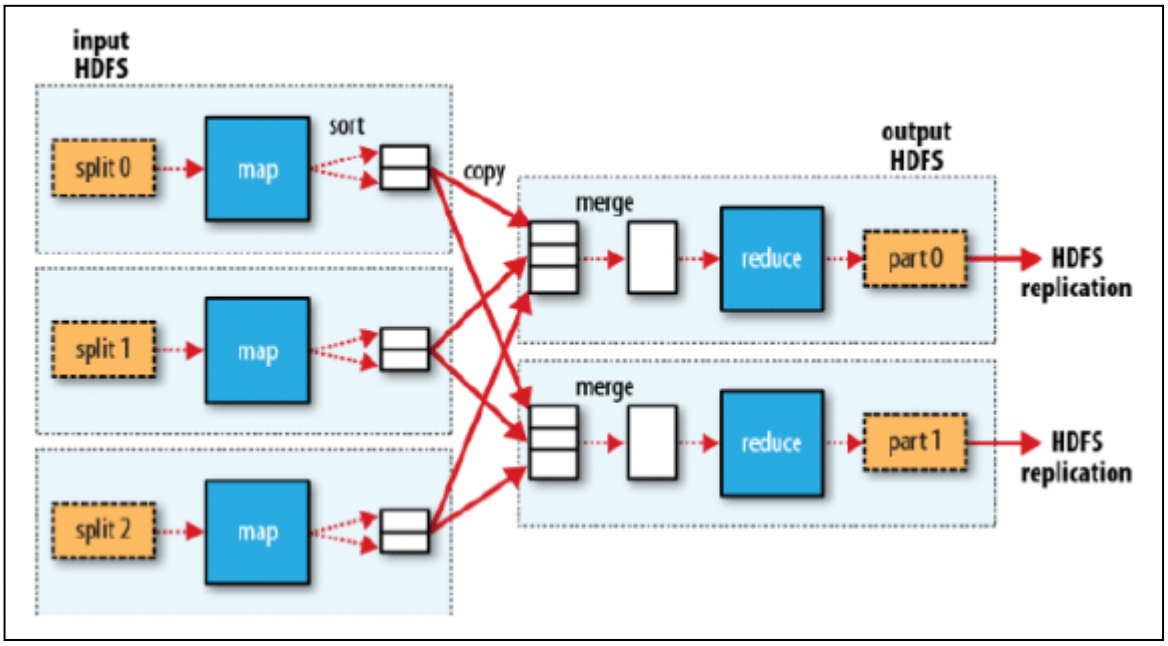

Figure 4. MapReduce Diagram.

As demonstrated, in the mapper.py and reducer.py code we get key-value from the mapper, and reducer uses the key-value pair to process the data.

```
 GNU nano 6.2                                  mapper.py
#!/usr/bin/env python3

import sys

#Read each line from stdin
for line in sys.stdin:

        # Get the words in each line
        words = line.split()

        # Generate the count for each word
        for word in words:

                # Write the key-value pair to stdout to be processed by
                # the reducer.
                # They key is anthing before the first tab character and the
                # value is anything after the first tab character.
                print('{0}\t{1}'.format(word,1))
```

Figure 5. Mapper code.

```
 GNU nano 6.2                                  reducer.py
#!/usr/bin/env python3

import sys

curr_word = None
curr_count = 0

# Process each key-value pair from the mapper
for line in sys.stdin:

        # Get the key and value from the current line
        word, count = line.split('\t')

        # Convert the count to an int
        count = int(count)

        # If the current word is the same as the previous word,
        # Increment its count, otherwise print the words count to stdout
        if word == curr_word:
                curr_count += count
        else:
                # Write word and its number of occurrences as key-value
                # pair to stdout
                if curr_word:
                        print('{0}\t{1}'.format(curr_word,curr_count))

                curr_word = word
                curr_count = count

# Output the count for the last word
if curr_word == word:
        print('{0}\t{1}'.format(curr_word, curr_count))
```

Figure 6. Reducer code.

As we can tell from the figures, the MapReducer works properly.

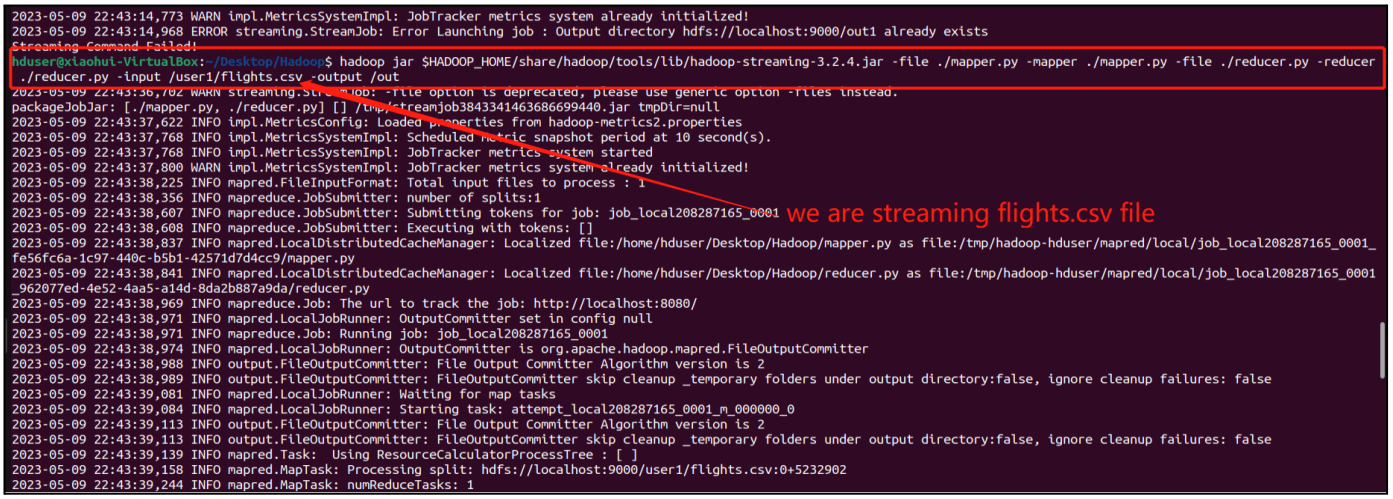

Figure 7. Initiation of MapReducer.

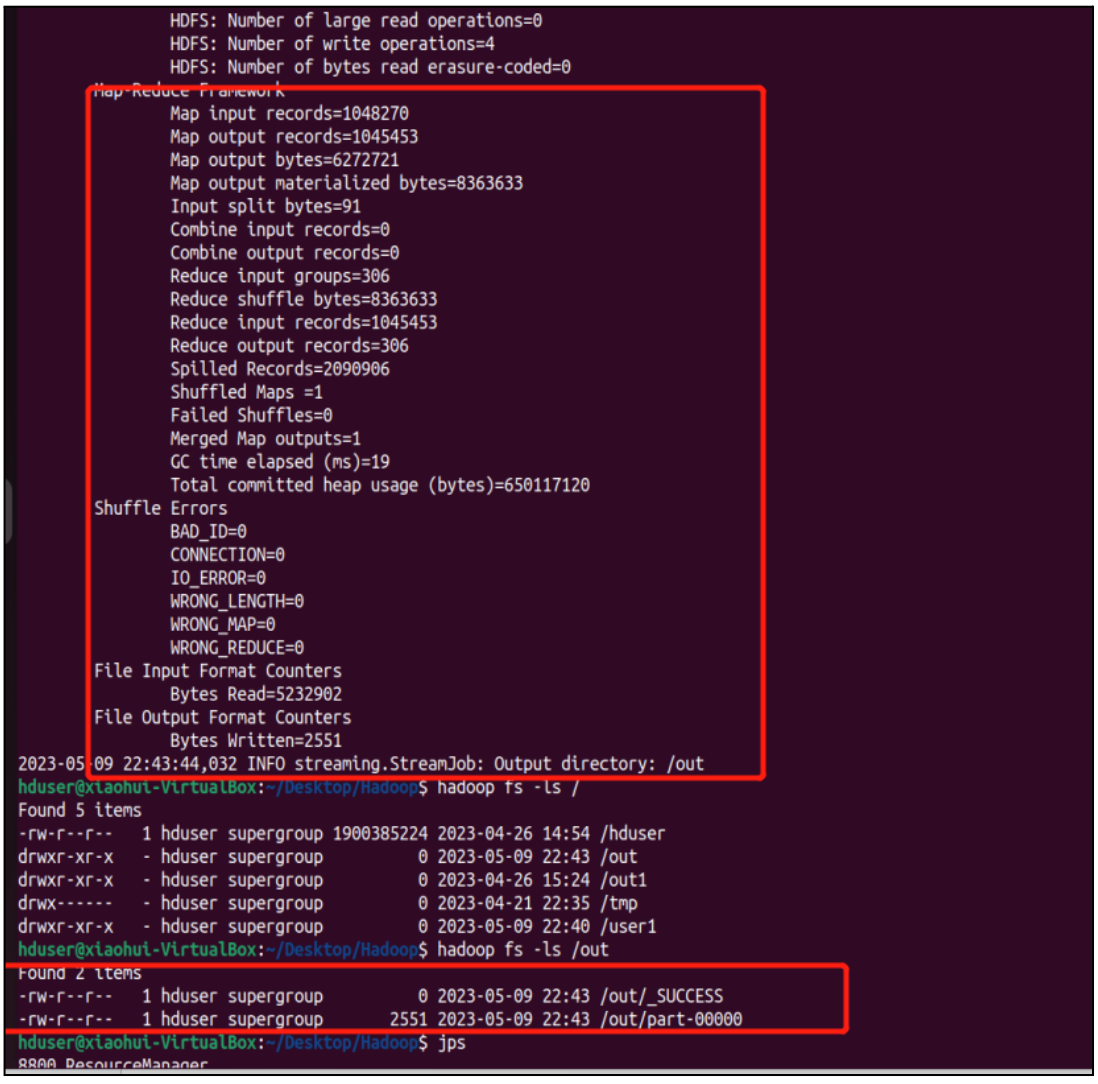

Figure 8. Results of MapReducer.

Architecture of the data storage

The structure of the database was designed by the Bureau of Transportation Statistics of the United States of America to keep track of the delays in the arrival and landing of internal flights within the period 2016-2018 and this data collection helped to estimate close to of a loss of almost 4 million dollars in the country's gross domestic product. The database has a collection Flights which contains date, time, origin, destination, airline, distance, and delay status of flights. This information was organised in a csv file that we were able to get from the Kaggle website. This data was imported into MongoDB which allowed us to query and review the data update that we were able to incorporate during this project.

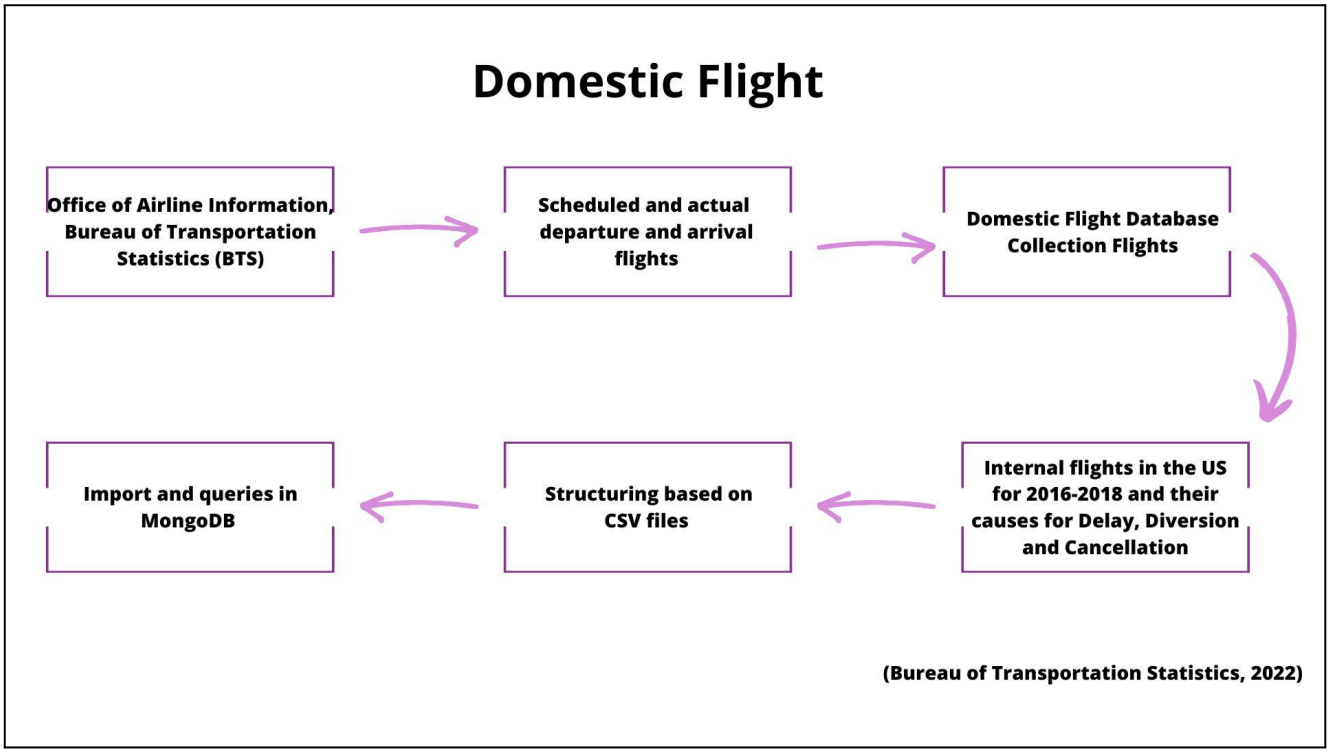

Figure 9. Data Storage Architecture Diagram.

The queries we made to verify that our CRUD program were as follows:

- Consult to find specific information:

  db.flights.find({ year: 2023 })

- Search for a specific data set, this helped us verify data created:

```
db.flights.find({
        year: 2023,
        month: 5,
        dayOfMonth: { $gte: 1, $lte: 10 }
})
```

- Query data in ascending order:

```
db.flights.find().sort({ arrTime: 1 })
```

Performance of NoSQL databases

As we know, there are many types of NoSQL databases, such as MongoDB, HBase, Redis, Memcached etc.. and YCSB (Yahoo! Cloud Serving Benchmark) which is one of the good tools to test the performance of NoSQL databases. It provides RunTime, Throughput, MinLatatency, MaxLatetency and so on. The test has a lot of detail. In this case we have tested the performance of MongoDB.

Figure 10. YCSB Test.

YCSB workloads A is 50% read and 50% Update, we could tell from the chart, the Runtime is 1275ms, and theThroughput is 784.3137254901961. So throughput can reach about 78.4%. The result returned is correct, 1000. 95thPercentieleLatency:665, and 99thPercentileLatency:1115.

The 5 output workloads.

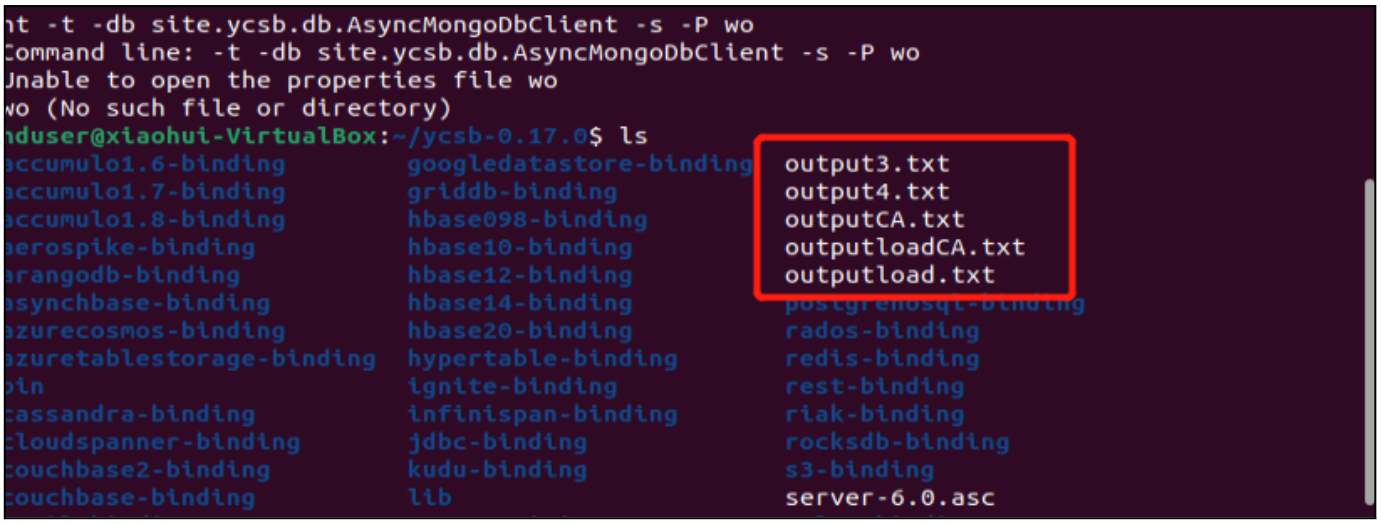

Figure 11. YCSB output files.

In mongoDB:

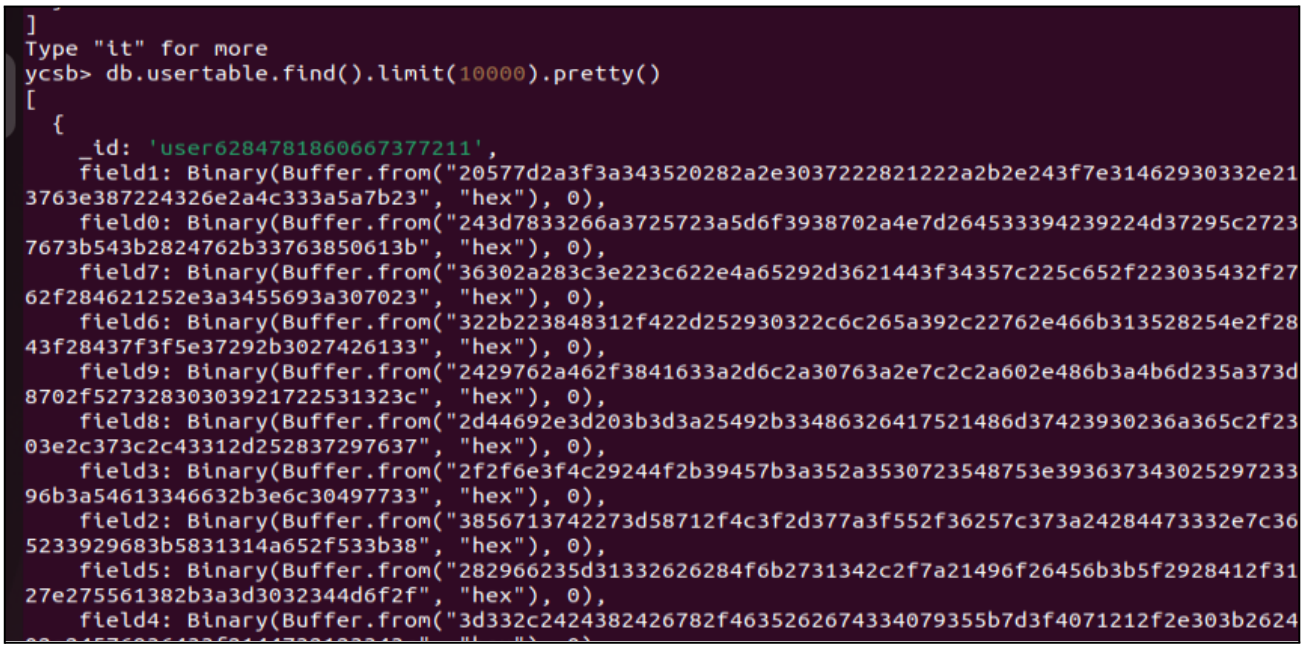

Figure 12. MongoDB interaction.

Spring Application

We have created a Web Application capable of connecting into a NoSQL (MongoDB) remote database and performing CRUD: CREATE – READ – UPDATE – DELETE methods.

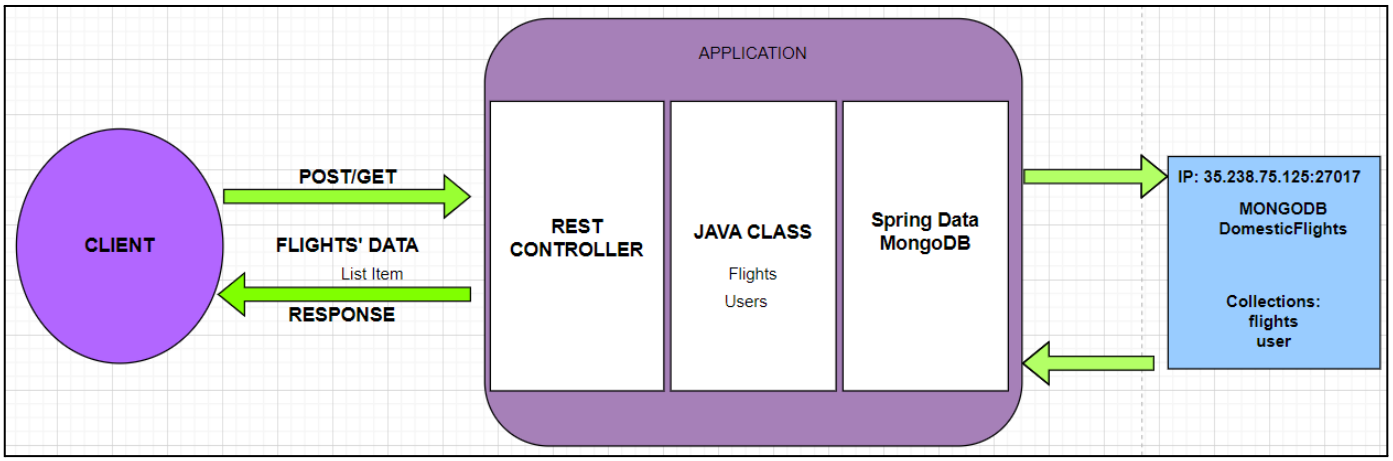

Figure 13. Spring application diagram.

In the diagram above we can find a brief explanation of how the spring application works in this project. A client can request data from a MongoDB based on Google Cloud passing by the REST controller that will provide the data required.

MongoDB

We opted to use a dedicated database instance on Google Cloud, thanks to the credits given by CCT to its students this semester. The mongoDB instance is a Linux machine running MongoDB standalone server 24/7. **The Instance will be UP for testing till June 14**.

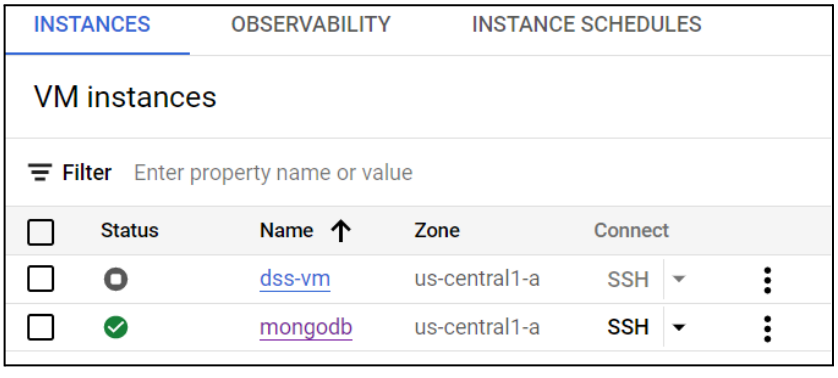

Figure 14. Linux Virtual Machine.

It was necessary to modify some firewall rules to allow access from anyone running the Web Application.

Figure 15. Linux Virtual Machine.

```
hduser@mongodb:~$ sudo systemctl status mongod.service
● mongod.service - MongoDB Database Server
     Loaded: loaded (/lib/systemd/system/mongod.service; enabled; vendor preset: enabled)
     Active: active (running) since Mon 2023-05-01 20:00:03 UTC; 1 weeks 1 days ago
       Docs: https://docs.mongodb.org/manual
   Main PID: 38869 (mongod)
     Memory: 918.9M
     CGroup: /system.slice/mongod.service
             └─38869 /usr/bin/mongod --config /etc/mongod.conf

May 01 20:00:03 mongodb systemd[1]: Started MongoDB Database Server.
hduser@mongodb:~$
```

Figure 16. MongoDB Server StandAlone: (IP 35.238.75.125).

```
DomesticFlights> db.flights.find()
[
  {
    _id: ObjectId("644ac12789d0961acb4089a7"),
    Year: 2016,
    Month: 1,
    DayofMonth: 17,
    DepTime: 1053,
    CRSDepTime: 1100,
    ArrTime: 1435,
    CRSArrTime: 1438,
    Origin: 'DFW',
    Dest: 'DTW',
    Distance: 986,
    _class: 'com.ca.IntegratedCA.model.Flights'
  },
  {
    _id: ObjectId("644ac12789d0961acb4089a8"),
    Year: 2016,
    Month: 1,
    DayofMonth: 17,
    DepTime: 1055,
    CRSDepTime: 1100,
    ArrTime: 1436,
    CRSArrTime: 1438,
    Origin: 'DFW',
    Dest: 'DTW',
    Distance: 986,
    _class: 'com.ca.IntegratedCA.model.Flights'
  },
  {
```

Figure 17. Database DomestiFlights collection flights.

The REST controller is composed of Methods POST and GET which control the workflow of the data sent and received either by the client or the database able to control multiple queries at the same time. However, the application has a login area to ensure that only authorized access can pass by to its system.

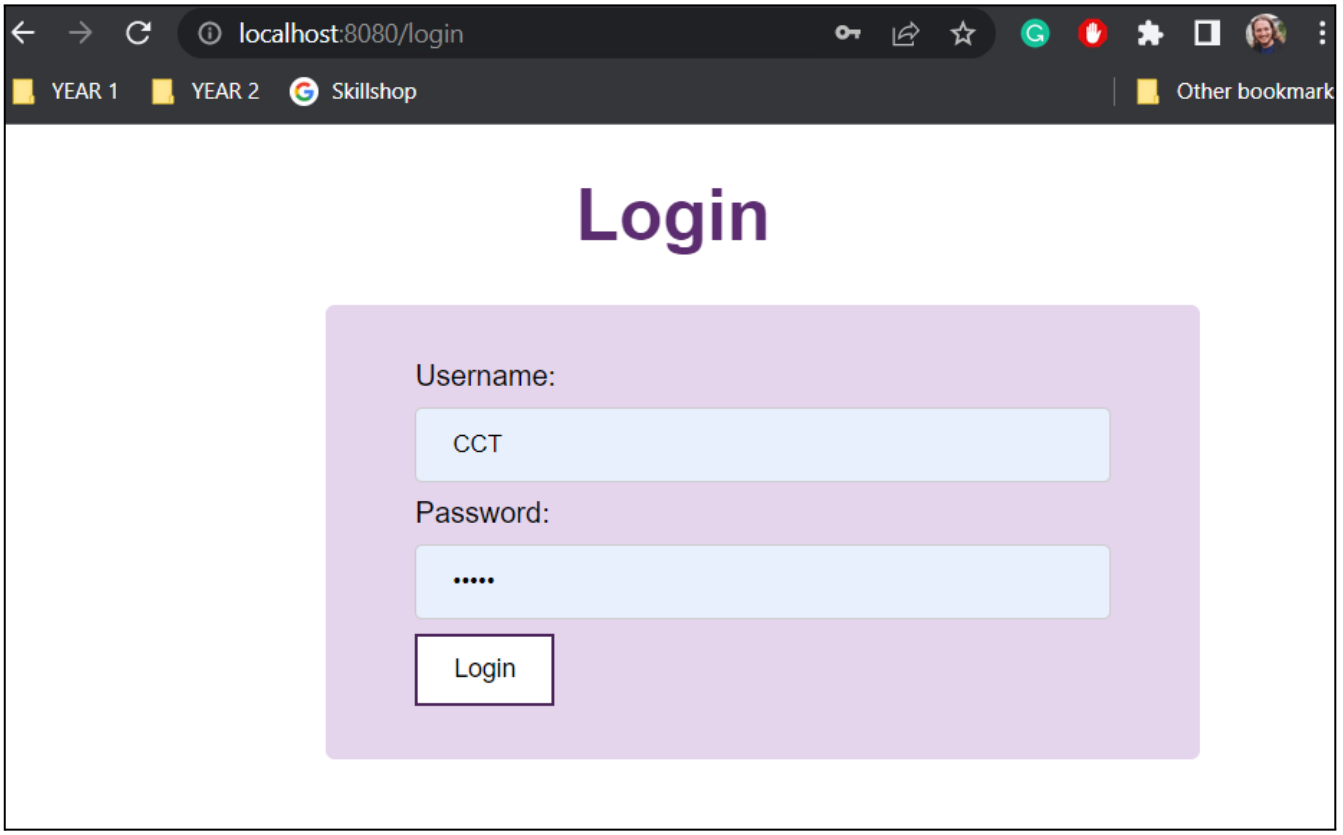

Figure 18. Login.html file.

Once the client access is granted it will be displayed in the index.html the first 10 pieces of information from the database Domestic Flights which contains over 16 million documents. As we can see in the picture below:

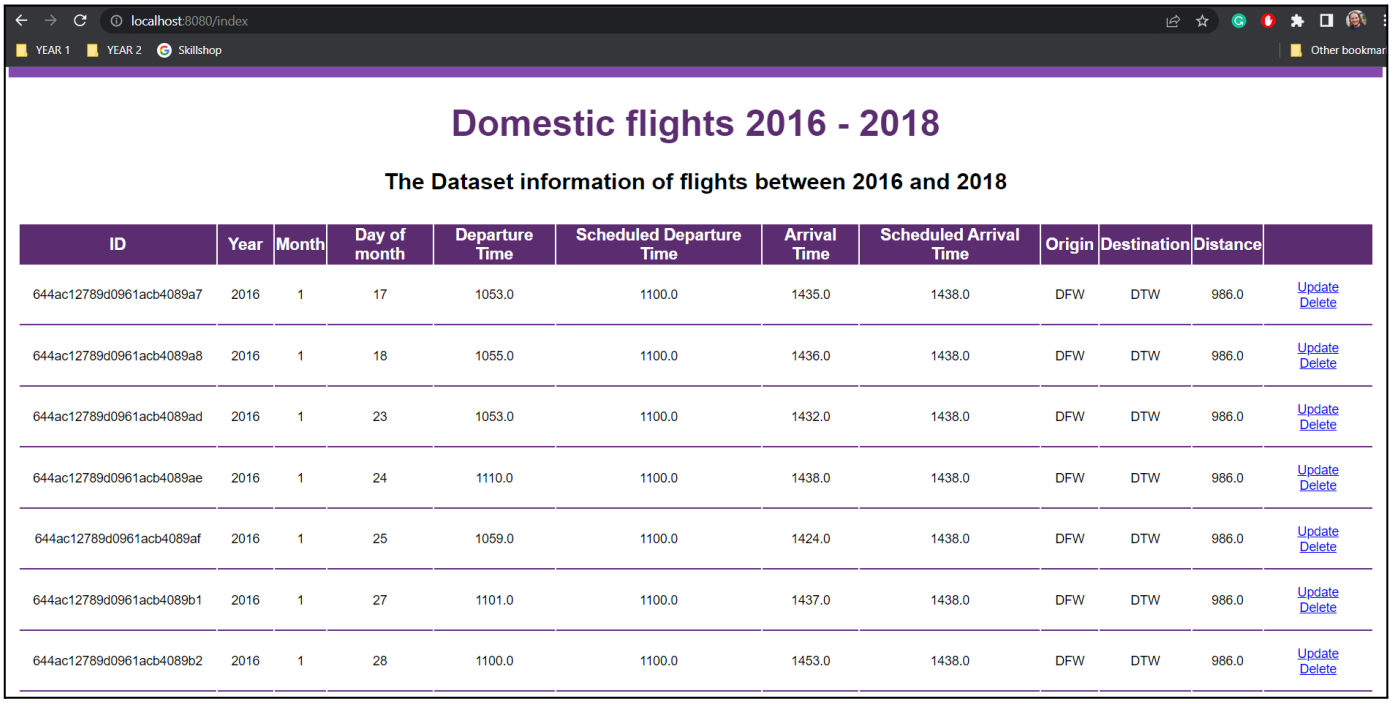

Figure 19. List of flights in index.html file.

On the right side, the client can Update the flight record as shown in the picture below:

Figure 20. Update and delete function.

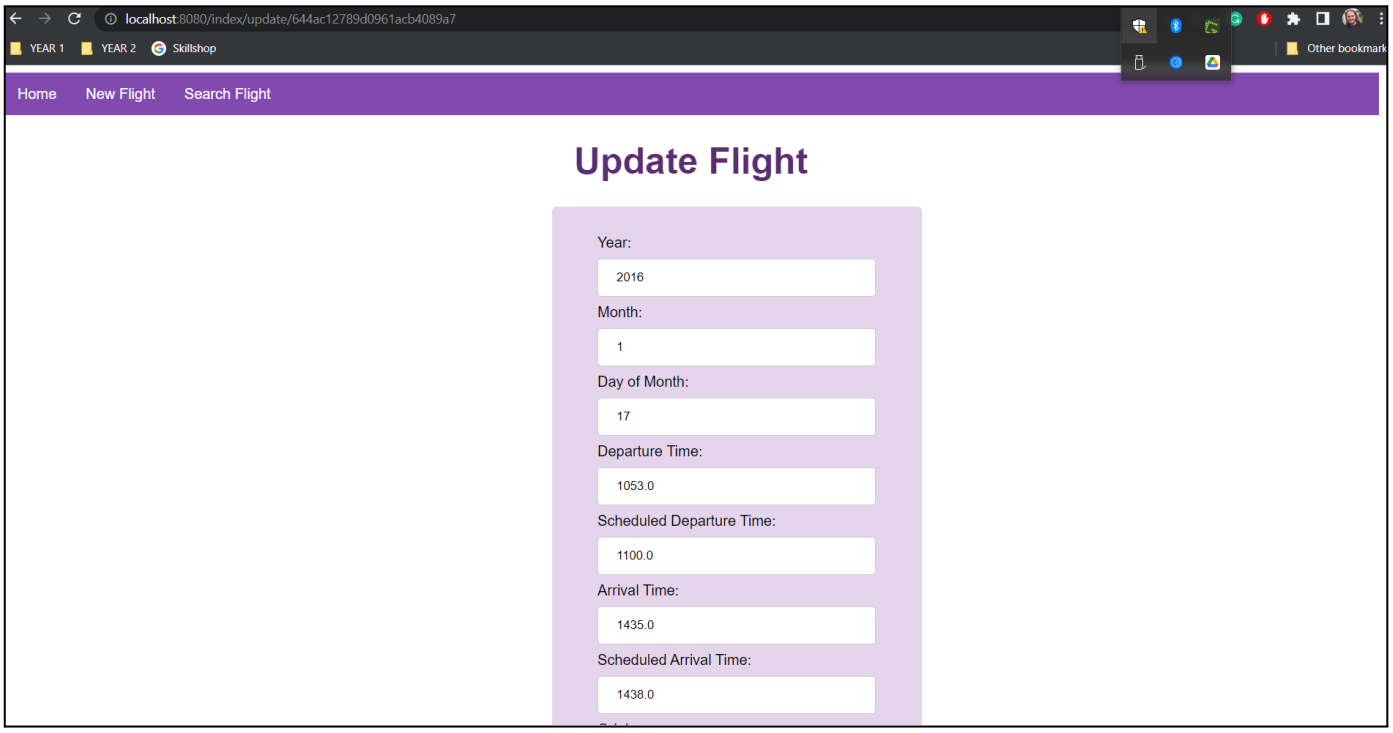

Figure 21. Update.html file.

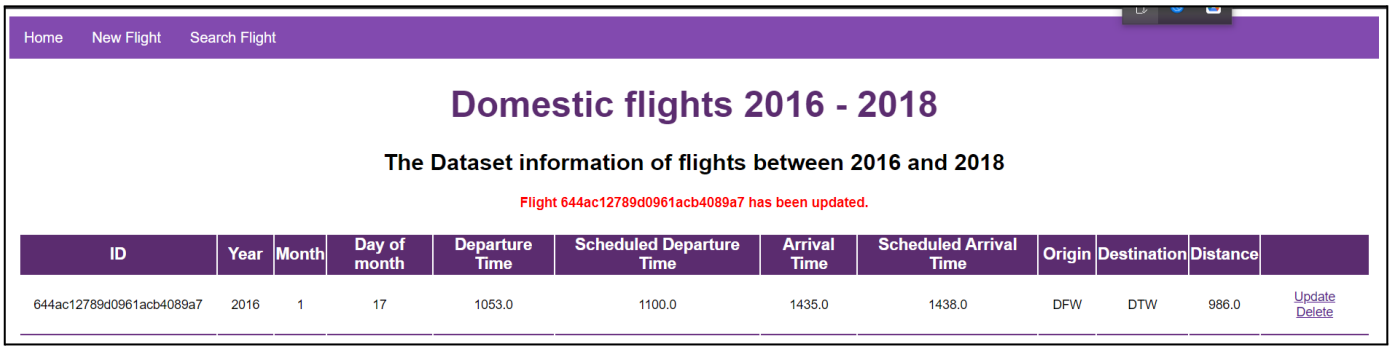

Figure 22. Notification update successfully.

Also, it is possible to add/create a new flight record just by clicking on the link at the top of the page “New Flight”:

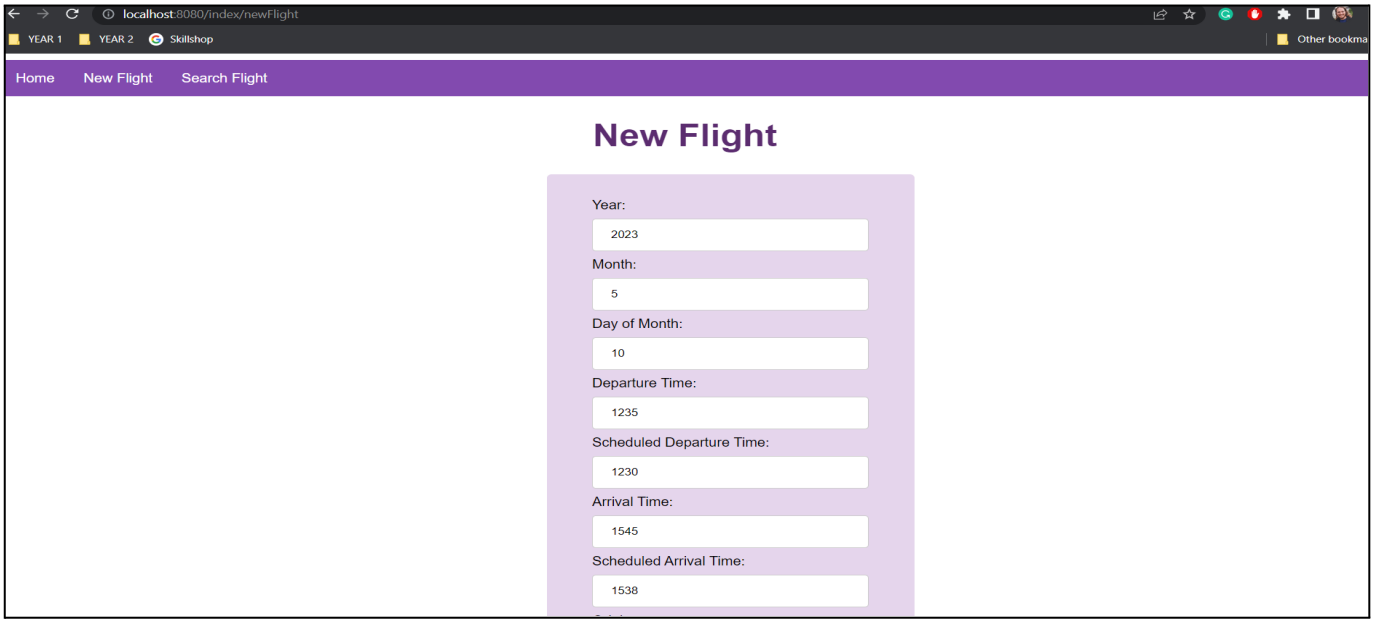

Figure 23. NewFlight.html file.

Returning the new ID created automatically by MongoDB :

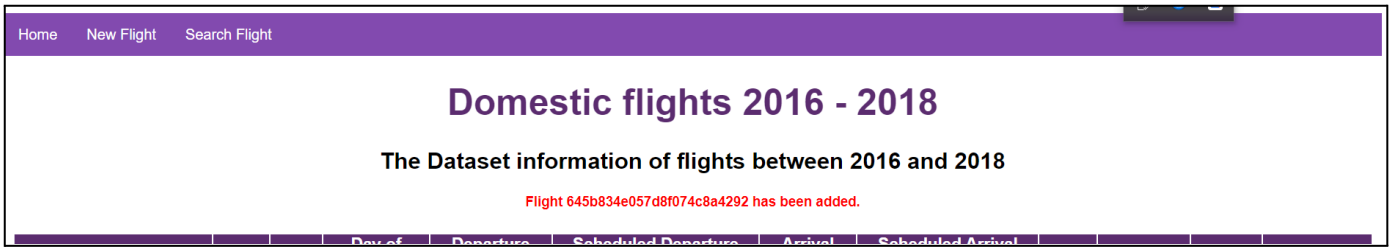

Figure 24. New flight was added successfully.

Using the ID as a reference the client can search for any flight information by clicking at the top of the page link “Search Flight”:

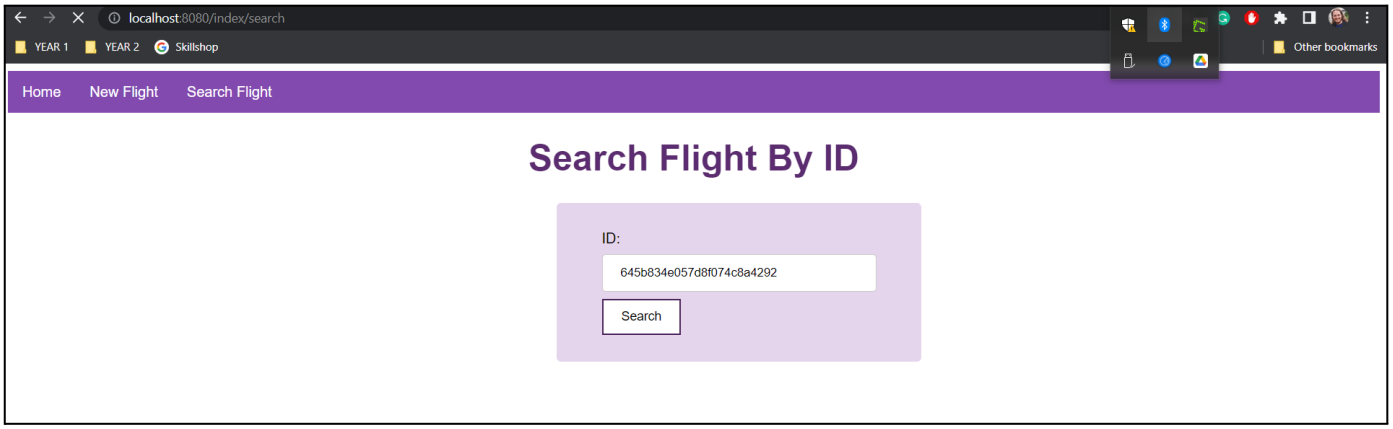

Figure 25. Search flight form.

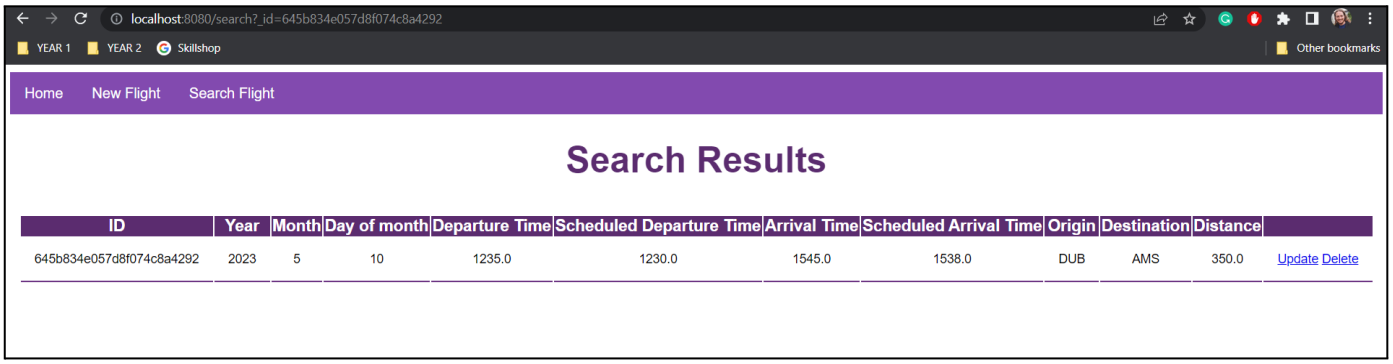

Figure 26. Results of searching flight.

Or delete a flight record:

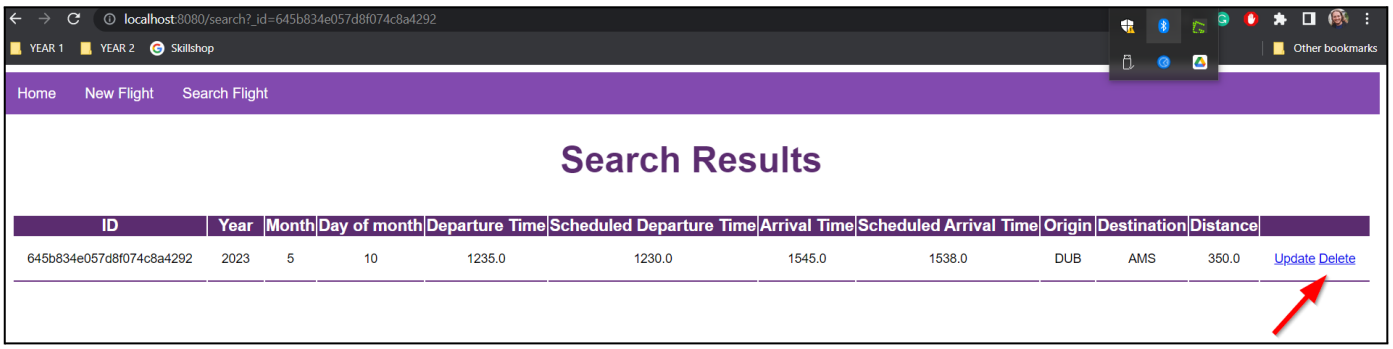

Figure 27. Delete flight function.

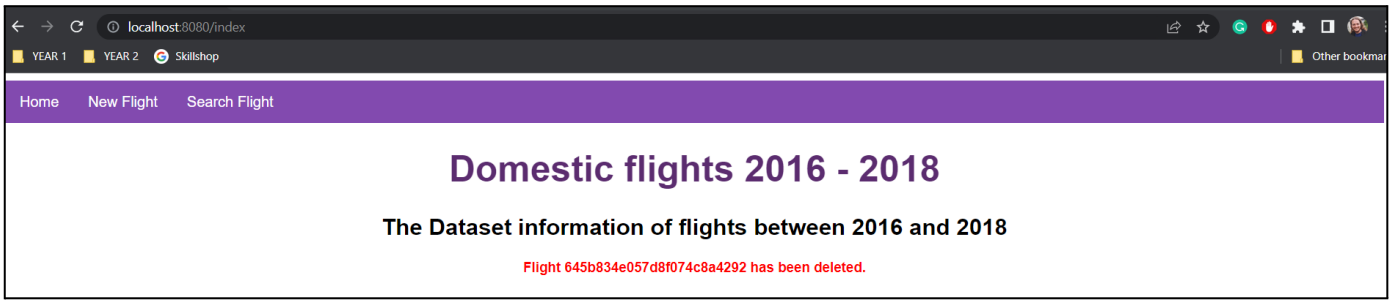

Figure 28. Notification was deleted successfully.

Here goes the core Java code the controller:

```
@Controller

public class FlightsController {

// LOGIN - MontoTemplate used for authentication
```

```
@Autowired

private MongoTemplate mongoTemplate;

// Add a new User to the model for authentication

@GetMapping({ "/login" })

public String loginForm(Model model) {

model.addAttribute("user", new User());

return "login";

}

// find the User in the database. If the user exists redirect to index page. Else, return to login page

@PostMapping("/login")

public String loginSubmit(@ModelAttribute User user, Model model, HttpSession session) {

Query query = new Query(

Criteria.where("username").is(user.getUsername()).and("password").is(user.getPassword()));

User authenticatedUser = mongoTemplate.findOne(query, User.class);

if (authenticatedUser != null) {

session.setAttribute("user", authenticatedUser);

return "redirect:/index";

} else {

model.addAttribute("loginError", "Invalid username or password");

return "login";

}

}

// FLIGHT DATASET - Flights Repository

@Autowired

private FlightsRepository repository;
```

```
public FlightsController(FlightsRepository repository) {

this.repository = repository;

}

/* Controller method for displaying the index page.

* Get all flights from the dataset and add to a flight list

* Display only 10 flights per page at the time*/

@GetMapping({ "/index", "/" })

public String home(Model model, @RequestParam(defaultValue = "0") int page) {

int pageSize = 10; // The number of items per page

Pageable pageable = PageRequest.of(page, pageSize);

Page<Flights> flights = repository.findAll(pageable);

model.addAttribute("flights", flights.getContent());

model.addAttribute("currentPage", page);

model.addAttribute("totalPages", flights.getTotalPages());

return "index";

}

/*Controller method that display the form for Add a new Flight data in the database */

@GetMapping("/index/newFlight")

public String formFlight(Model model) {

Flights flights = new Flights();

model.addAttribute("flights", flights);

return "newFlight";

}

/*Save the new flights data to the database

* Returns an message confirming the flight has been created/added*/
```

```
@PostMapping("/index")
public String saveFlights(@ModelAttribute("flights") Flights flights, RedirectAttributes redirectAttributes) {
repository.save(flights);
redirectAttributes.addFlashAttribute("AddedMessage", "Flight " + flights.get_id() + " has been added.");
return "redirect:/index";
}
/*Controller method for Updating the data using an ID as reference*/
@GetMapping("/index/update/{_id}")
public String updateAllFlightById(@PathVariable String _id, Model model) {
model.addAttribute("flights", repository.findById(_id).orElse(null));
return "update";
}
/*Retrieves the existing flight data from the database then update all the properties
* Returns an message confirming the flight has been updated*/
@PostMapping("/index/{_id}")
public String updateFlight(@PathVariable String _id, @ModelAttribute("flights") Flights flight, Model model,
RedirectAttributes redirectAttributes) {
Flights existingFlight = repository.findById(_id).get();
existingFlight.setYear(flight.getYear());
existingFlight.setMonth(flight.getMonth());
existingFlight.setDayofMonth(flight.getDayofMonth());
existingFlight.setDepTime(flight.getDepTime());
existingFlight.setCRSDepTime(flight.getCRSDepTime());
existingFlight.setArrTime(flight.getArrTime());
existingFlight.setCRSArrTime(flight.getCRSArrTime());
```

```
existingFlight.setOrigin(flight.getOrigin());

existingFlight.setDest(flight.getDest());

repository.save(existingFlight);

redirectAttributes.addFlashAttribute("updateMessage", "Flight " + _id + " has been updated.");

return "redirect:/index";

}

/*Gets the request and creates a new Flights object to the model returing the search method*/

@GetMapping("/index/search")

public String searchFlight(Model model) {

Flights flights = new Flights();

model.addAttribute("flights", flights);

return "search";

}

/*Controller method for Searching the data using an ID as reference

* return an message if flight with the given ID was not found*/

@GetMapping("/search")

public String search(Model model, @RequestParam String _id, RedirectAttributes redirectAttributes) {

Flights flight = repository.findById(_id).orElse(null);

if (flight == null) {

redirectAttributes.addFlashAttribute("searchMessage", "Flight " + _id + " cannot be found!");

return "redirect:/index";

}

Flights flights = new Flights();

model.addAttribute("flight", flight);

model.addAttribute("flights", flights);
```

```
return "searchResults";

}

/* Method for deleting the flight data by ID*/

@GetMapping("/index/{_id}")

public String deleteFlightById(@PathVariable("_id") String _id, RedirectAttributes redirectAttributes) {

repository.deleteById(_id);

redirectAttributes.addFlashAttribute("deleteMessage", "Flight " + _id + " has been deleted.");

return "redirect:/index";

}

}
```

Contribution

At the beginning of our project, we had divided the tasks into 3 individual parts, however, after the first week of working on it we realized that the only way to make it was if we all worked together helping each other and as such the best way to have the work done. As you can see our first approach ended up delaying our delivery of the CA on time. Thus, we all worked together from the choice and deployment of the dataset through the utilisation of the processing environment to the development of the Web Application in Spring. The chart bar below represents the effort and time spent in the CA.

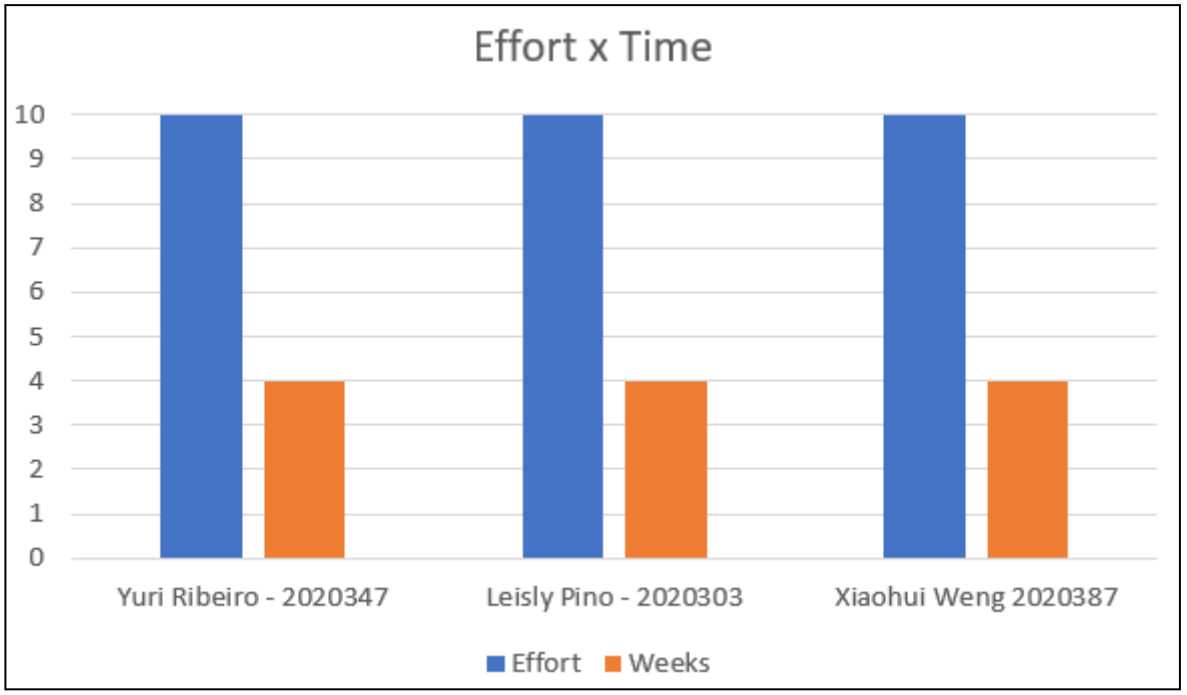

Figure 29. Team contribution diagram.

References

Bureau of Transportation Statistics, (2022). Domestic Flights 2016 - 2018 [online]. Kaggle: Your Machine Learning and Data Science Community. Available at: https://www.kaggle.com/datasets/rulyjanuarfachmi/domesticusairflight2016-2018

Spring Boot Thymeleaf CRUD example - BezKoder [online], (2023). BezKoder. Available at: https://www.bezkoder.com/spring-boot-thymeleaf-example/

Spring Data, Spring Boot, MongoDB (Example & Tutorial) [online], (2016). Tests4Geeks. Available at: https://tests4geeks.com/blog/spring-data-boot-mongodb-example/