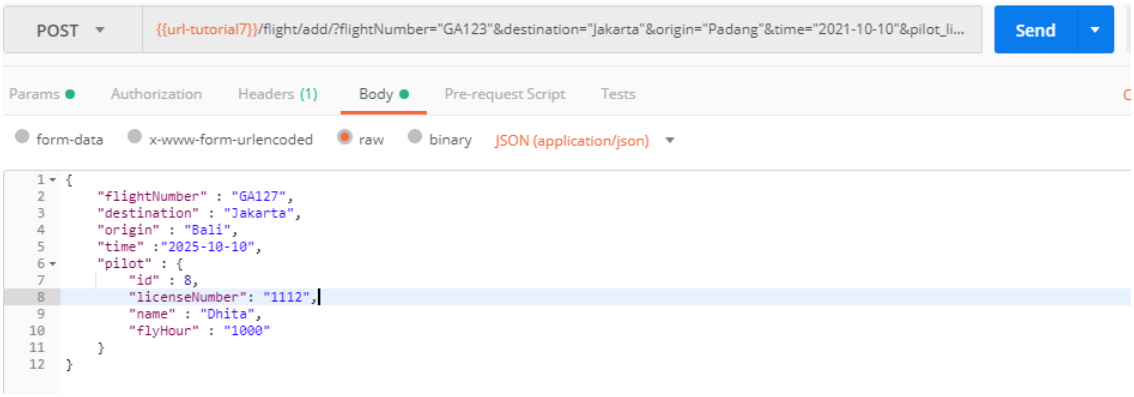
CHANADYA
1606917821
APAP-C

1. Buatlah FlightController menjadi RestController
   a. POST add flight
      Pada FlightController ditambahkan method seperti ini:

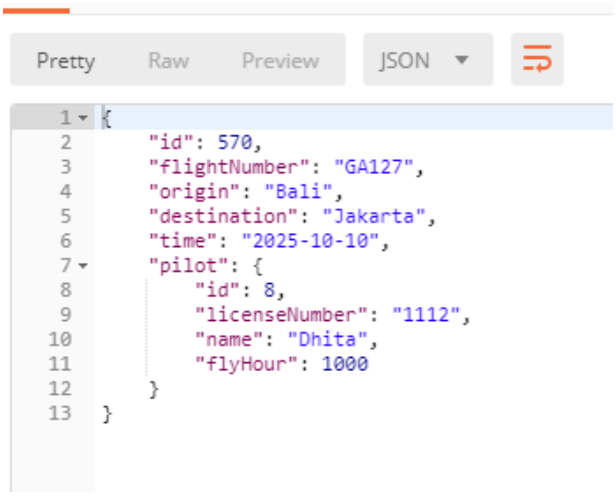
```
@PostMapping(value = "/add")
public FlightModel addFlightSubmit(@RequestBody FlightModel flight) {
    return flightService.addFlight(flight);
}
```

Penggunaan anotasi @PostMapping bertujuan untuk tipe request yang bertipe POST seperti pada method penambahan flight. Method addFlightSubmit tidak membutuhkan parameter atau path variable untuk url nya, namun membutuhkan request body berupa object dari entity FlightModel. Object tersebut yang nantinya akan ditambahkan pada database. Pada flightService telah tersedia method untuk menambahkan object pada database(flightDb). Method addFlightSubmit ini akan mereturn(response) object FlightModel yang ditambahkan tadi.

Pada saat dijalankan di PostMan, penambahan flight membutuhkan licenseNumber dari pilot yang bertanggung jawab untuk flight tersebut.



Pada saat send request, response akan mengembalikan object dari flight tersebut beserta id yang digenerate secara otomatis.



CHANADYA
1606917821
APAP-C

b. PUT update flight

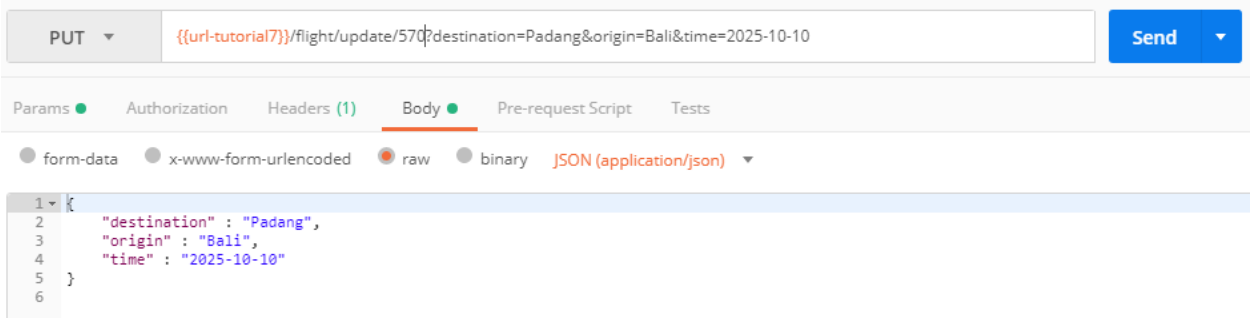
Pada FlightController ditambahkan method seperti dibawah ini:

```
@PutMapping(value = "/update/{flightId}")
public String updateFlightSubmit(@PathVariable("flightId") long flightId,
        @RequestParam(value = "destination", required = false) String destination,
        @RequestParam(value = "origin", required = false) String origin,
        @RequestParam(value = "time", required = false) Date time) {
    FlightModel flight = flightService.getFlight(flightId).get();
    if(flight.equals(null)) {
        return "Couldn't find your flight";
    }

    flight.setDestination(destination);
    flight.setOrigin(origin);
    flight.setTime(time);
    flightService.updateFlight(flightId, flight);

    return "Flight update success";
}
```

Penggunaan anotasi @PutMapping bertujuan untuk tipe request yang bertipe PUT seperti pada method meng*update flight*. Pada url untuk update flight dibutuhkan id dari flight tersebut, serta dibutuhkan parameter-parameter seperti 'destination', 'origin', dan 'time'. Parameter-parameter tersebut yang nanti akan diupdate di database.

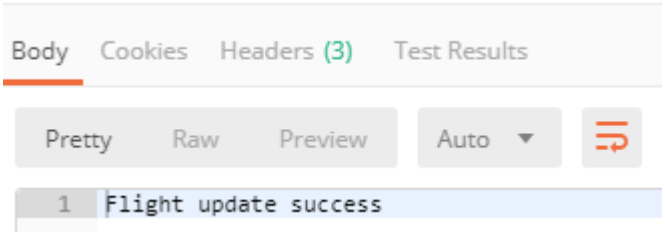
FlightId yang diperoleh dari url akan digunakan untuk mencari dan meretrieve object dengan id tersebut dari database. Apabila tidak ada flight yang sesuai dengan id yang diminta, maka method tersebut akan mereturn "Couldn't find your flight". Apabila terdapat flight dengan id yang diminta pada database, maka destination, origin,dan time dari flight tersebut akan diupdate. Untuk mengupdate data yang ada di database, maka dilakukan pemanggilan method pada flightService yang membutuhkan id dari flight yang akan diupdate dan object yang akan diupdate.

Pada postman dilakukan send request seperti berikut:



Flight berhasil diupdate:

CHANADYA
1606917821
APAP-C



c. GET flight

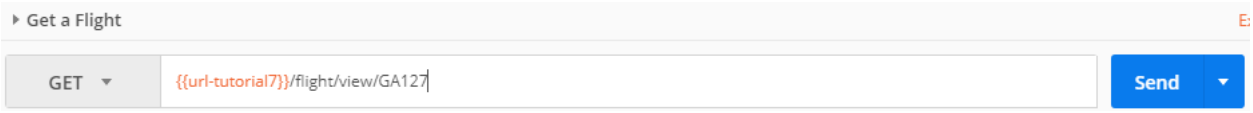
Pada FlightController ditambahkan method seperti berikut:

```
@GetMapping(value = "/view/{flightNumber}")
public FlightModel flightView(@PathVariable("flightNumber") String flightNumber) {
    FlightModel flight = flightService.findFlightByFlightNumber(flightNumber).get();
    return flight;
}
```

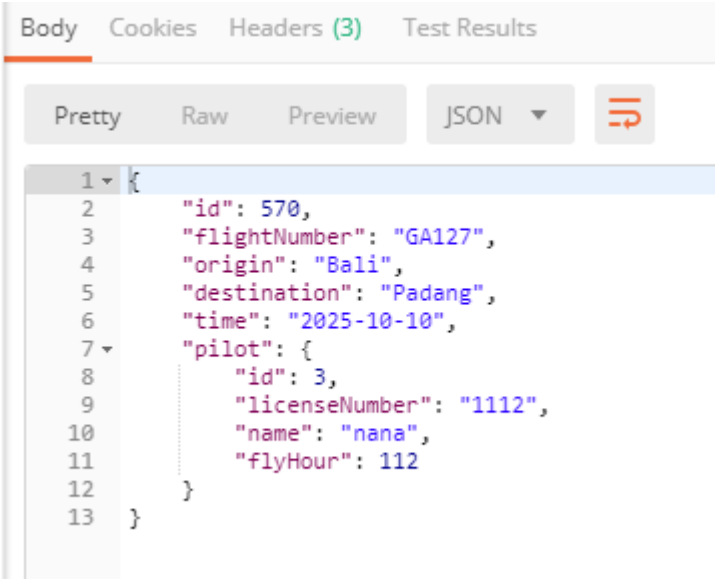
Penggunaan anotasi @GetMapping bertujuan untuk tipe request yang bertipe GET seperti pada method menampilkan flight tertentu.

Method ini membutuhkan data berupa flightNumber yang akan digunakan untuk meretrieve data dari database. Response dari method ini adalah flight dengan flightNumber yang diminta.

Pada postman dilakukan send request seperti berikut:



Response dari request tersebur adalah object Flight yang diminta:



d. GET all flight

Pada FlightController ditambahkan method seperti berikut:

CHANADYA
1606917821
APAP-C

```
@RequestMapping (value = "/all")
private List<FlightModel> viewAllFlight () {
    List <FlightModel> allFlights = flightService.allFlight();
    return allFlights;
}
```

Method tersebut akan mengembalikan semua object Flight yang ada pada database dengan tipe List.
Dilakukan send request seperti berikut:

Response dari request tersebut adalah List flight yang ada pada database:

e. Delete flight
Pada FlightController ditambahkan method seperti berikut:

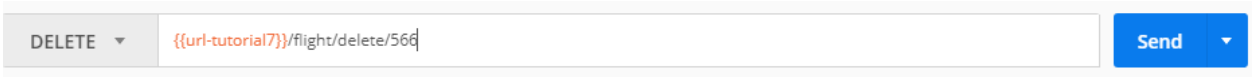
```
@DeleteMapping(value="/delete/{flightId}")
public String deleteFlight(@PathVariable("flightId") long flightId) {
    FlightModel flight = flightService.getFlight(flightId).get();
    flightService.deleteFlight(flight);
    return "success";


}
```

Penggunaan anotasi @DeleteMapping bertujuan untuk tipe request yang bertipe DEL seperti pada method penghapusan flight.

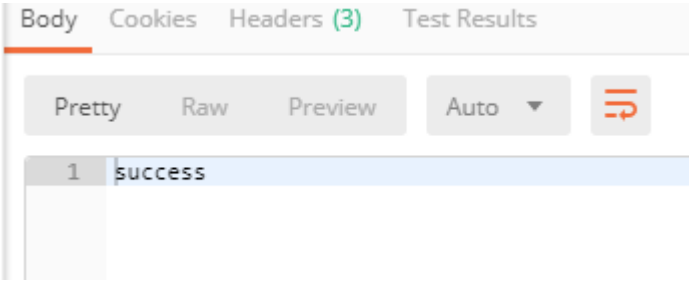
CHANADYA
1606917821
APAP-C

Method ini akan mereturn string “success” apabila data berhasil dihapus dari database. Method ini membutuhkan id dari flight yang akan dihapus. Id tersebut akan digunakan untuk meretrieve object flight dari database, yang kemudian akan dihapus.

Pada postman akan dilakukan send request seperti berikut:



Apabila data berhasil dihapus, maka responsenya adalah sebagai berikut:



2. Pada latihan nomor 2, langkah pertama yang dilakukan adalah membuat akun pada Amadeus untuk memperoleh API yang nantinya akan digunakan untuk menampilkan data airport di suatu kota di Indonesia. Pada latihan ini Jakarta dijadikan sebagai contoh.

Name	Values	Description
apikey *(required)*	HbU2HYmLeO9BTAQ2B	API Key provided for your account, to identify you for API access. Make sure to keep this API key secret.
term *(required)*	Jakarta	Search keyword that should represent the start of a word in a city or airport name.
country	ID	Identified a country based of a ISO 3166-1 alpha-2 code
all_airports	true	Boolean to include or not all airports, no matter their traffic rank. False by default, to only display relevant airports.

Berikut API yang diperoleh:

Request Response cURL

```
curl -X GET "https://api.sandbox.amadeus.com/v1.2/airports/autocomplete?apikey=HbU2HYmLeO9BTAQ2B6kFIyuhnJel
2qqA&term=Jakarta&country=ID&all_airports=true"
```

API ini nanti akan digunakan pada Setting.java sebagai airportUrl.

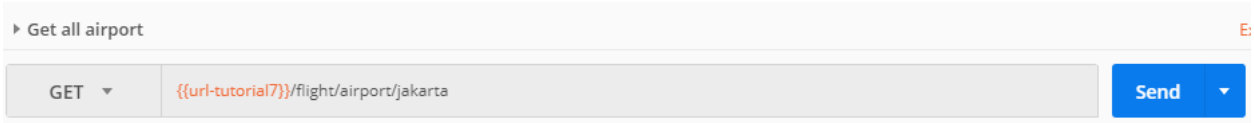
CHANADYA
1606917821
APAP-C

```
FlightContr...  PilotControl...  PilotServic...  PilotDetail....  Setting.java  AirportDetai...  »14
1  package com.apap.tutorial7.rest;
2
3  public class Setting {
4      final public static String pilotUrl = "https://e16ed5f0-98bf-4618-9f5e-6ae8381dd0f7.mock.pstmn.io";
5      final public static String airportUrl = "https://api.sandbox.amadeus.com/v1.2/airports/autocomplete?apikey=HbU2HYmLe
6  }
7
```

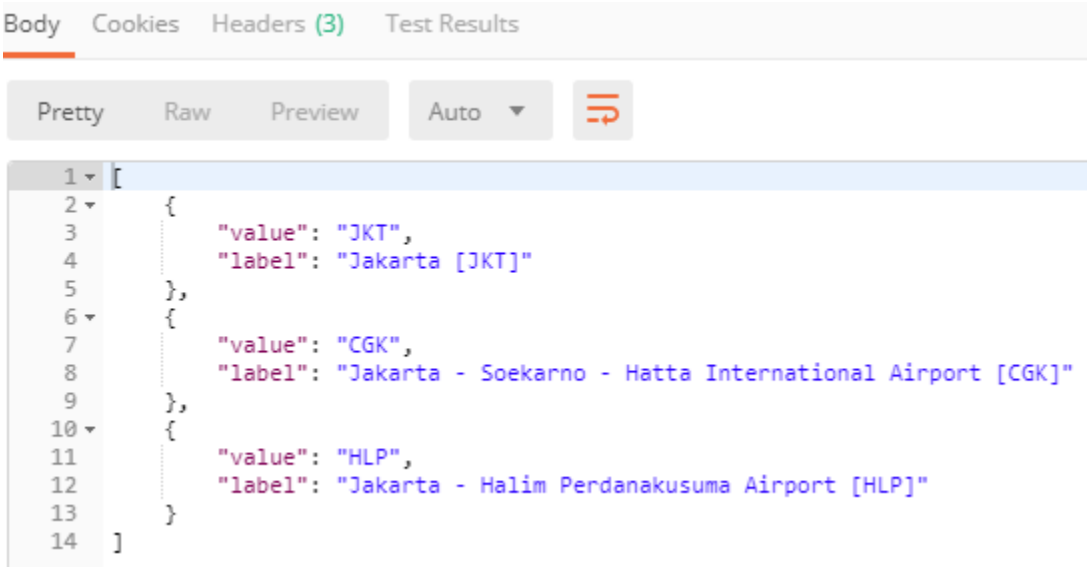
Pada FlightController ditambahkan method getAirport yang membutuhkan cityTerm(Jakarta) pada urlnya.

```
@Autowired
RestTemplate restTemplateAirport;


@Bean
public RestTemplate restAirport() {
    return new RestTemplate();
}

@GetMapping(value="/airport/{cityTerm}")
public String getAirport(@PathVariable("cityTerm") String cityTerm) throws Exception{
    String path = Setting.airportUrl + cityTerm;
    return restTemplateAirport.getForEntity(path, String.class).getBody();
}
```

Pada postman dilakukan send request seperti berikut:



Response dari request tersebut adalah :




```
1  [
2      {
3          "value": "JKT",
4          "label": "Jakarta [JKT]"
5      },
6      {
7          "value": "CGK",
8          "label": "Jakarta - Soekarno - Hatta International Airport [CGK]"
9      },
10     {
11         "value": "HLP",
12         "label": "Jakarta - Halim Perdanakusuma Airport [HLP]"
13     }
14 ]
```