Tutorial 7

1. Web Service

Web Service adalah sekumpulan application logic beserta object dan method yang dimilikinya yang terletak di suatu server yang terhubung ke internet sehingga dapat diakses menggunakan protokol HTTP dan SOAP (Simple Object Access Protocol). Web service memberikan layanan untuk melakukan komunikasi antara dua atau lebih mesin melalui jaringan. Web service dibangun di atas beberapa teknologi web seperti XML, SOAP, WSDL, dan UDDI. Terdapat perbedaan antara web service dengan API. Web service bukan merupakan open source tetapi bisa digunakan oleh semua client yang memahami XML. Sedangkan, API merupakan open source dan hanya bisa digunakan oleh semua client yang memahami JSON atau XML.

2. Service Producer

Service Provider memberikan layanan untuk memberi dan mengubah data kepada client yang digunakan melalui anotasi RestController.

3. Service Consumer

Service Consumer memberikan layanan data kepada client dengan bantuan dari data lain untuk memenuhi kebutuhan client. Pada dasarnya service consumer membutuhkan data lain sebelum bisa memberikan data ke client. Service consumer memberikan layanan untuk membuat mockserver yang merupakan server bayangan API yang bisa diakses dengan spesifikasi custom. Mockserver adalah service yang ada di Postman.

Latihan

1. Membuat FlightController menjadi RestController dengan spesifikasi tertentu.

Method : POST add flight

Pada FlightModel, terdapat atribut pilot yang memiliki anotasi JsonIgnore.

```
@ManyToOne(fetch = FetchType.LAZY, optional = false)
@JoinColumn(name = "pilot_licenseNumber", referencedColumnName = "license_number")
@OnDelete(action = OnDeleteAction.CASCADE)
@JsonIgnore
private PilotModel pilot;
```

Hapus anotasi tersebut seperti gambar di bawah untuk mendapatkan licenseNumber dari pilot, karena jika ada JsonIgnore maka field tersebut akan diignore sementara.

```
@ManyToOne(fetch = FetchType.LAZY, optional = false)
@JoinColumn(name = "pilot_licenseNumber", referencedColumnName = "license_number")
@OnDelete(action = OnDeleteAction.CASCADE)

private PilotModel pilot;
```

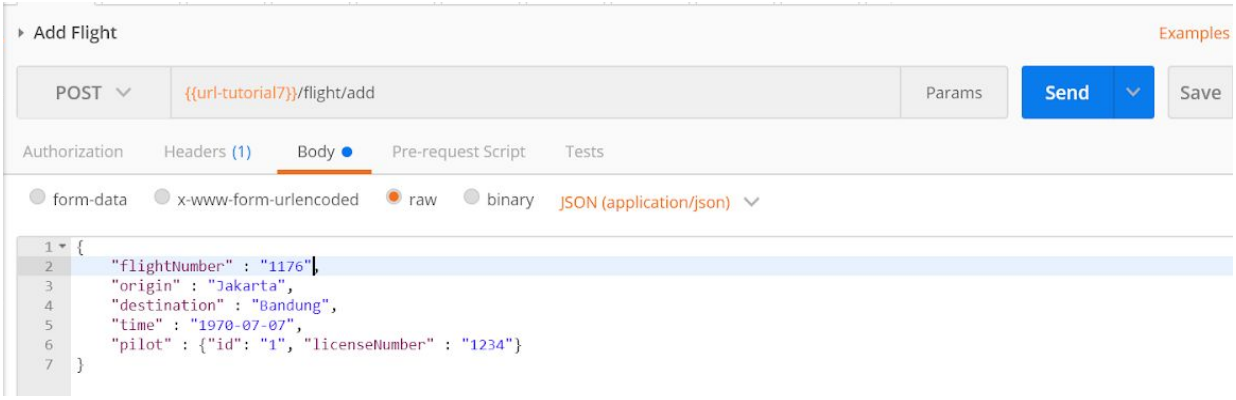
Kemudian, tambahkan anotasi JsonIgnore ke PilotModel seperti di bawah ini. Hal ini untuk mengabaikan listFlight yang ada di PilotModel.

```
@OneToMany(mappedBy = "pilot", fetch = FetchType.LAZY)
@JsonIgnore
private List<FlightModel> listFlight = new ArrayList<FlightModel>();
```

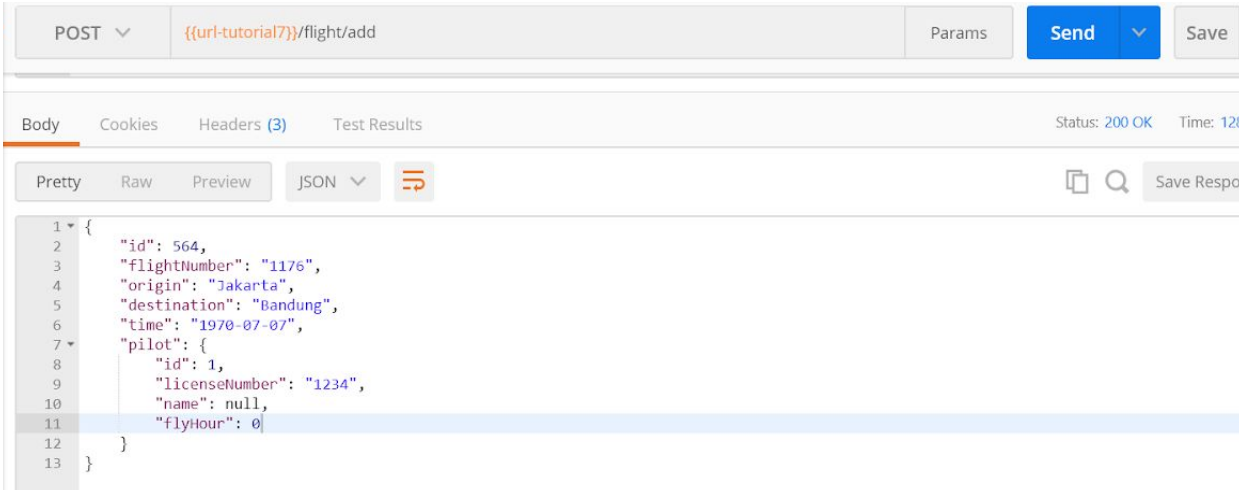
Pada controller, tambahkan method sebagai berikut.

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

Setelah itu, buat request pada Postman dan beri nama **Add Flight** seperti di bawah ini. Isi body sesuai dengan kebutuhan.



Klik Save dan Send. Kemudian akan muncul response yang dikembalikan seperti di bawah ini.



Method : PUT update flight

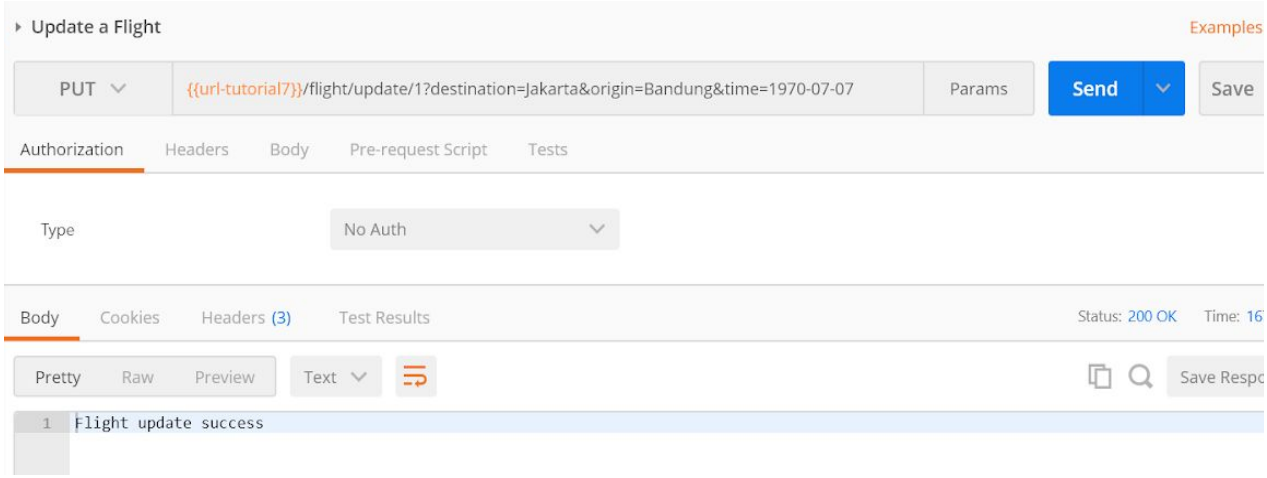
Implementasikan method updateFlight pada FlightService dan FlightServiceImpl. Kemudian, tambahkan method di controller sebagai berikut.

```
@PutMapping(value = "/update/{flightId}")
public String updateFlightSubmit(@PathVariable("flightId") long flightId,
        @RequestParam("destination") String destination,
        @RequestParam("origin") String origin,
        @RequestParam("time") Date time) {
    FlightModel flight = flightService.getFlightDetailByFlightId(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";
}
```

Buat request pada Postman dan beri nama **Update a Flight** seperti di bawah ini. Isi parameter sesuai dengan kebutuhan.

Klik Save dan Send. Kemudian akan muncul response yang dikembalikan seperti di bawah ini.

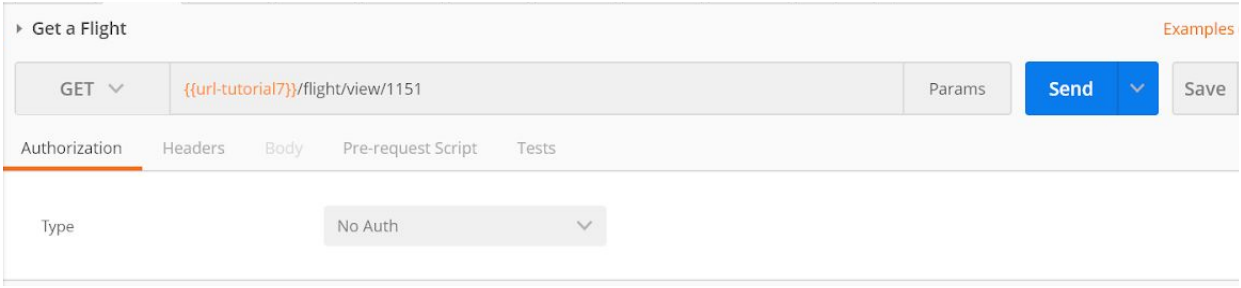


Method : GET flight

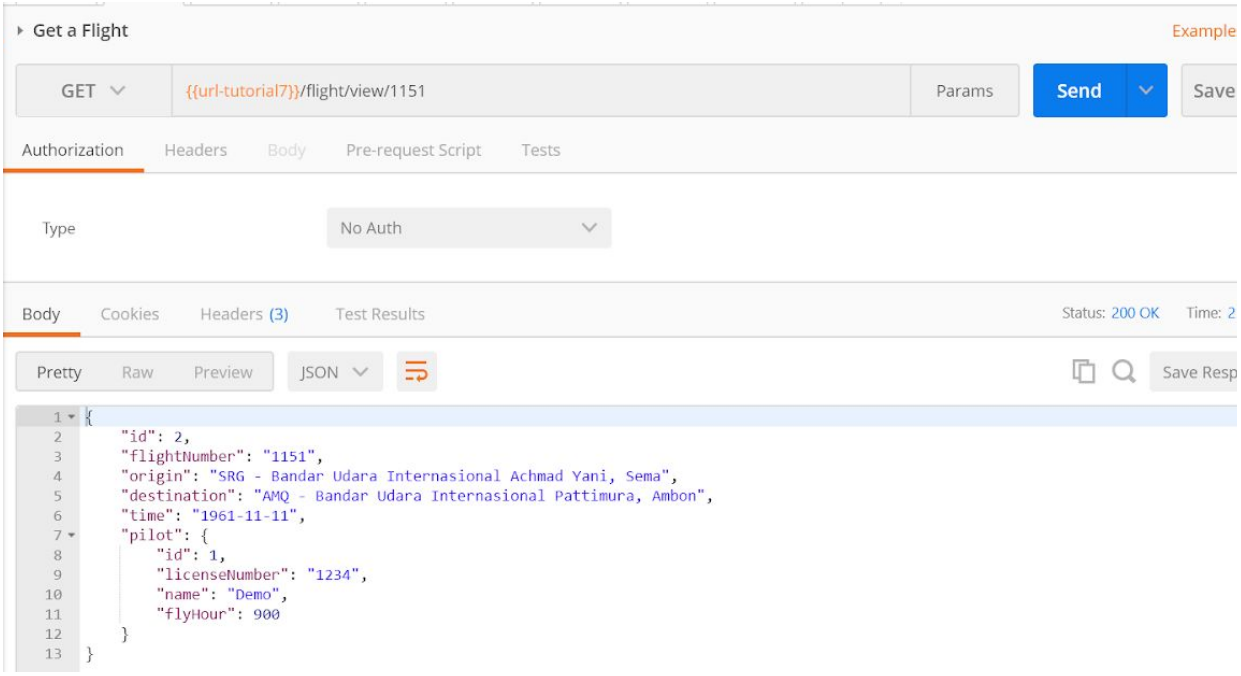
Pada controller, tambahkan method sebagai berikut.

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

Buat request pada Postman dan beri nama **Get a Flight** seperti di bawah ini.



Klik Save dan Send. Kemudian akan muncul response yang dikembalikan seperti di bawah ini.

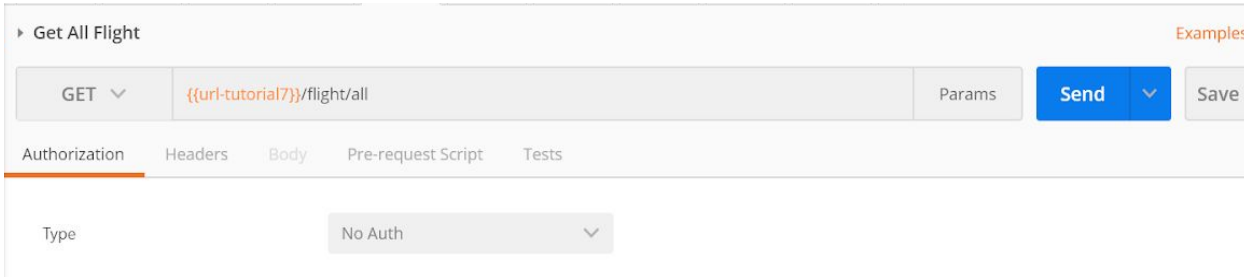


Method : GET all flight

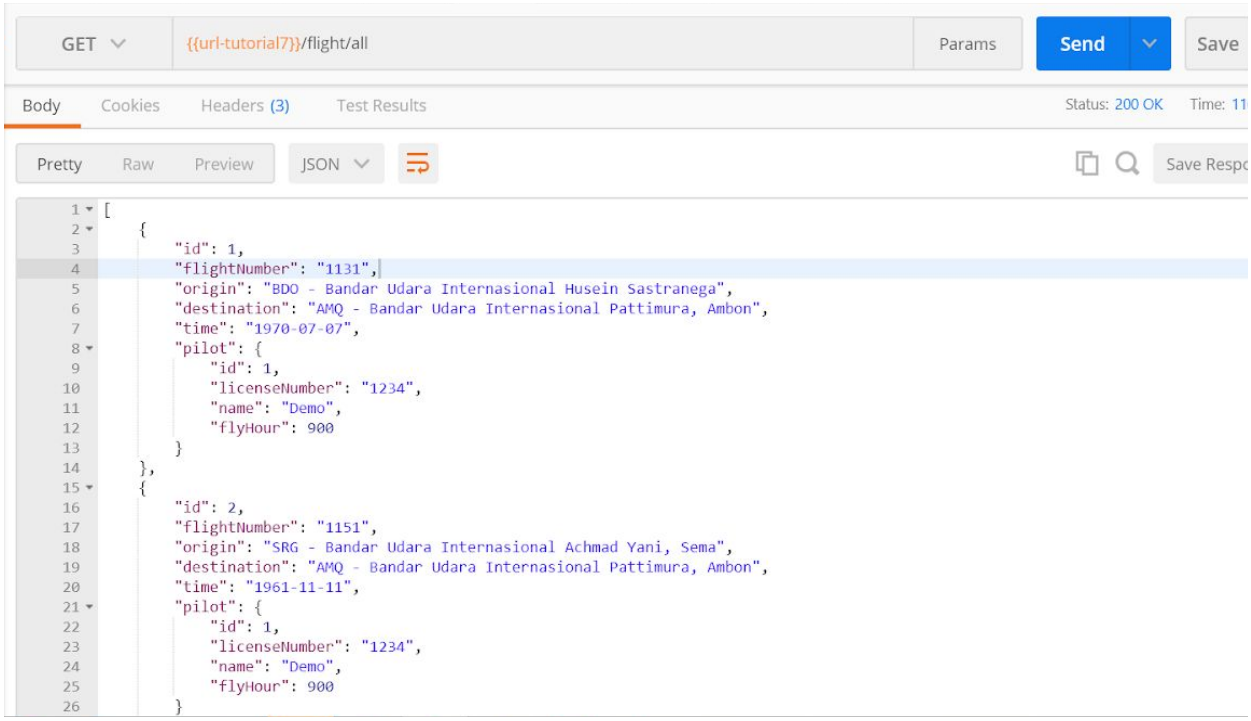
Pada controller, tambahkan method sebagai berikut.

```
@GetMapping(value = "/all")
public List<FlightModel> flightAll() {
    List<FlightModel> allFlight = flightService.getAllFlight();
    return allFlight;
}
```

Buat request pada Postman dan beri nama **Get All Flight** seperti di bawah ini.



Klik Save dan Send. Kemudian akan muncul response yang dikembalikan seperti di bawah ini.



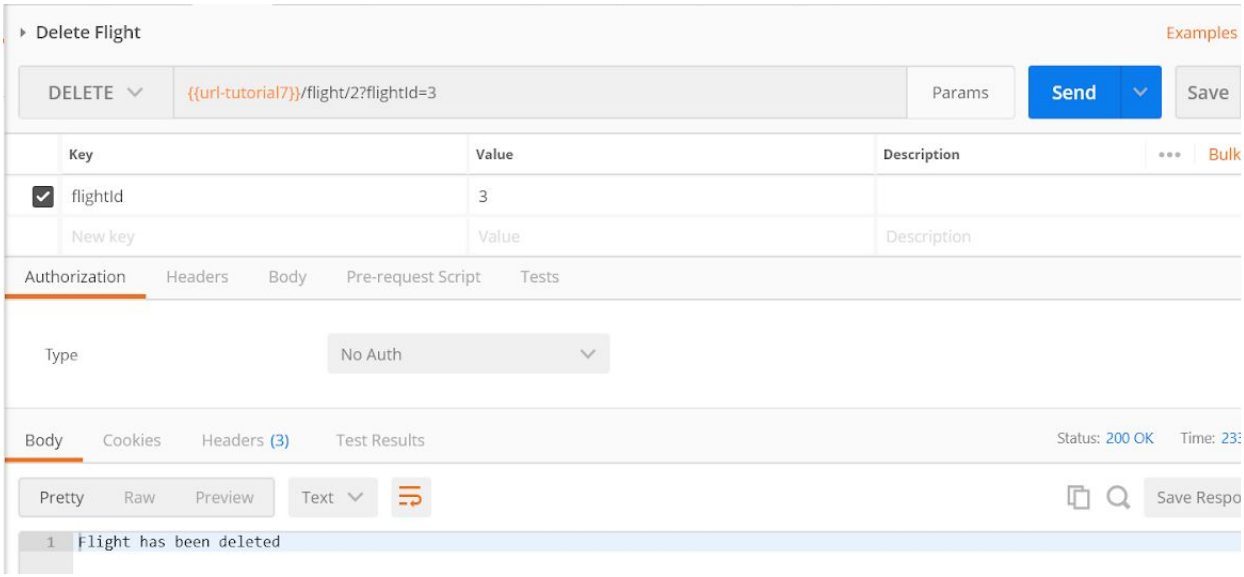
Method : DELETE flight

Implementasikan method deleteFlight pada FlightService dan FlightServiceImpl. Pada controller, tambahkan method sebagai berikut.

```
@DeleteMapping(value = "/{flightId}")
private String deleteFlight(@RequestParam("flightId") long flightId) {
    FlightModel flight = flightService.getFlightDetailByFlightId(flightId).get();
    flightService.deleteFlight(flight);
    return "Flight has been deleted";
}
```

Buat request pada Postman dan beri nama **Delete Flight** seperti di bawah ini. Isi parameter sesuai dengan kebutuhan.

Klik Save dan Send. Kemudian akan muncul response yang dikembalikan seperti di bawah ini.



2. Sebelum mendapatkan API key, buat akun di Amadeus dan buat sebuah app yang akan menyimpan consumer key. Kemudian, isi Query Parameters dengan consumer key yang sudah didapatkan untuk mendapatkan API key. Jika sudah, salin URL yang diterima ke dalam file Setting.java seperti di bawah ini.

```
final public static String airportUrl = "https://api.sandbox.amadeus.com/v1.2/airports/autocomplete?"
        + "apikey=40P383mj3Y34t0xeAkLdODyERf80Gv17&country=ID&all_airports=true";
```

Kemudian, buat sebuah class baru untuk mendefinisikan detail dari airport pada package rest. Setelah itu, buat class baru pada package controller dengan method untuk mengimplementasikan airport controller.

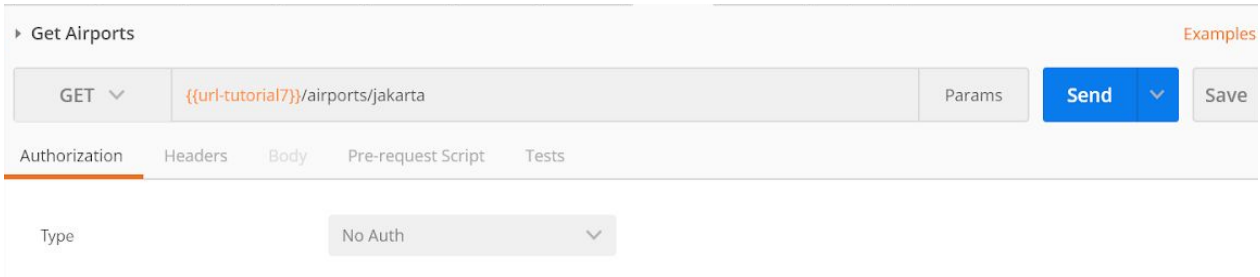
```
@RestController
public class AirportController {

    @Autowired
    private RestTemplate restTemplate;

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

    @GetMapping(value="/airports/{city}")
    public String getAirportsByCity(@PathVariable("city") String city) throws Exception{
        String path = Setting.airportUrl + "&term="+city;
        return restTemplate.getForEntity(path, String.class).getBody();
    }
```

Buat folder baru di Postman dengan nama Airport untuk menyimpan request baru tentang airport. Kemudian, buat request baru dengan nama **Get Airports** seperti di bawah ini.



Klik Save dan Send. Kemudian akan muncul response yang dikembalikan seperti di bawah ini.