Latihan 1

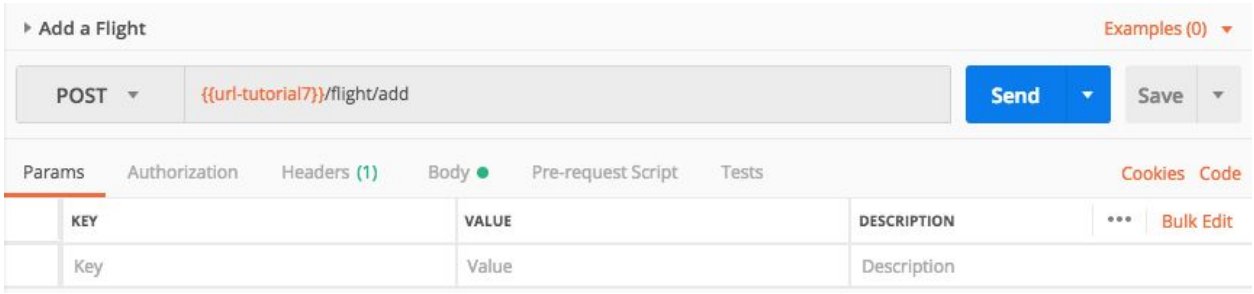
1. Menambahkan flight

Pada *controller* implementasikan method addFlight seperti berikut:

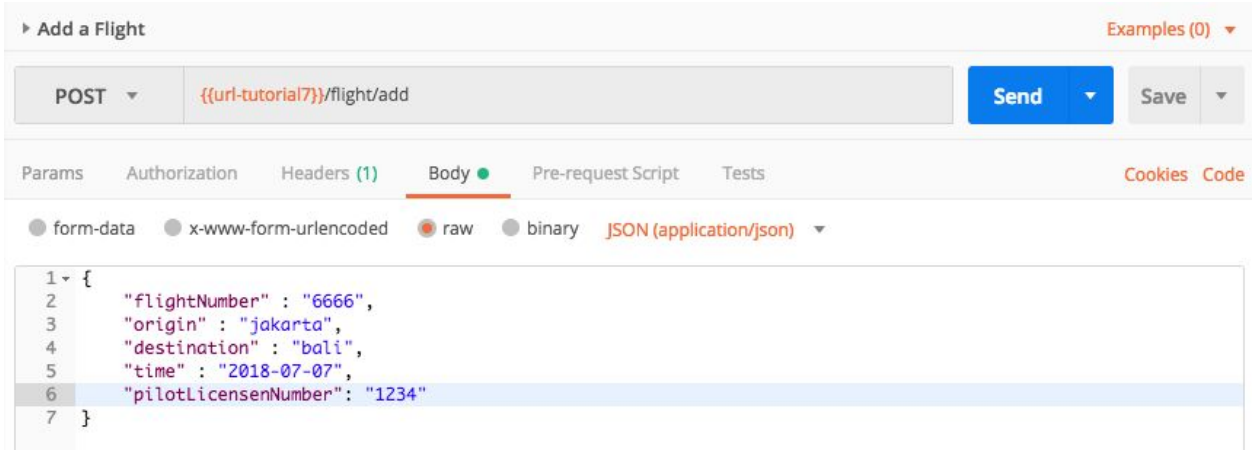
Gunakan anotasi **@PostMapping,** anotasi ini merupakan versi simplifikasi dari **@RequestMapping(method = RequestMethod.POST).**

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

Lalu pada postman kirim *request POST:*



Kemudian tambahkan *requestbody* pada tab body dengan tipe JSON



Send *request* maka akan dikembalikan response sebagai berikut*:*

//belom

2. Update flight

Implementasikan method updateFlight pada flight service:

```
@Override
public void updateFlight(FlightModel flight) {
    flightDb.save(flight);
}
```

Implementasikan method updateFlight pada *controller* sebagai berikut:
Gunakan anotasi **@PutMapping,** anotasi ini merupakan versi simplifikasi dari **@RequestMapping(method = RequestMethod.PUT).**

```
@PutMapping(value = "/update/{flightId}")
public String updateFlight(@PathVariable("flightId") long flightId,
        @RequestParam(value = "destination") String destination,
        @RequestParam(value = "origin") String origin,
        @RequestParam(value = "time") Date time) {
    FlightModel flight = flightService.getFlighttDetailById(flightId).get();
    if (flight.equals(null)) {
        return "couldn't find the flight";
    }
    flight.setDestination(destination);
    flight.setOrigin(origin);
    flight.setTime(time);
    flightService.updateFlight(flight);
    return "flight update success";

}
```

Setelah itu kirim *request PUT* pada postman

3. View flight

Implementasikan method flightView pada controller sebagai berikut:
Gunakan anotasi **@GetMapping,** anotasi ini merupakan versi simplifikasi dari **@RequestMapping(method = RequestMethod.GET).**

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

Kirim *request GET* pada postman dengan menginput *flightNumber* yang ada pada database

Send *request* maka akan dikembalikan response sebagai berikut

4. View all flight

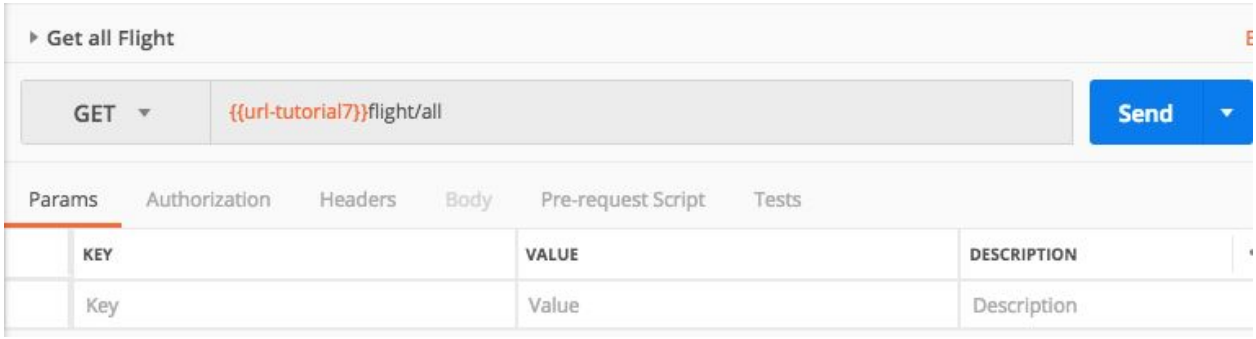
Untuk dapat mengembalikan list flight, implementasikan method getAllFlight pada flightService:

```
@Override
public List<FlightModel> getAllFlight() {
    return flightDb.findAll();
}
```

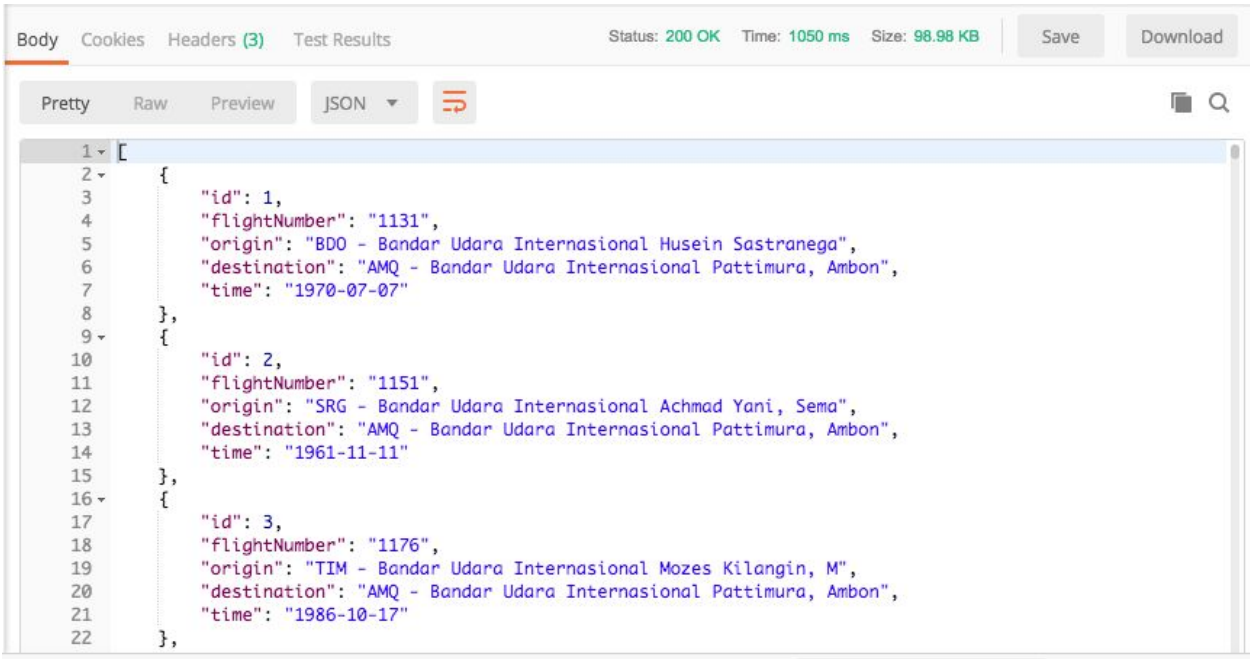
Lalu implementasikan method viewAllFlight pada *controller:*

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

Send *request* pada postman



Maka akan dikembalikan response sebagai berikut:



5. Delete flight

Implementasikan method delete pada flightService

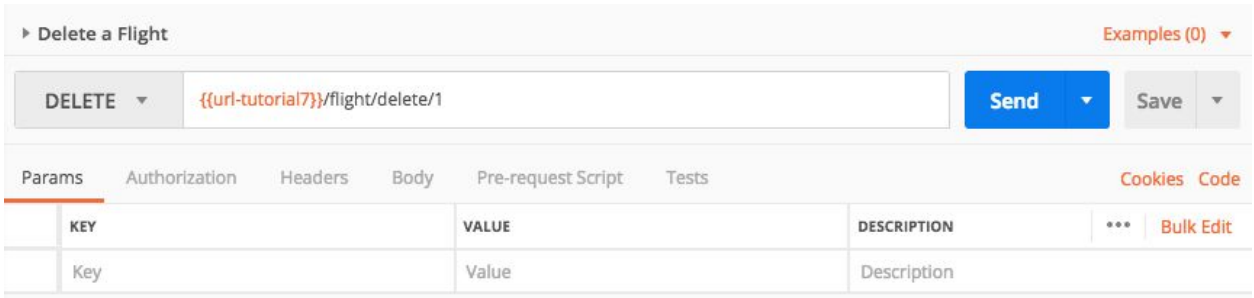
```
@Override
public void delete(FlightModel flight) {
    flightDb.delete(flight);
}
```

Lalu implementasikan method deleteFlight pada controller:

Gunakan anotasi **@DeleteMapping,** anotasi ini merupakan versi simplifikasi dari **@RequestMapping(method = RequestMethod.DELETE).**

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

Kirimkan request delete pada postman



Jika berhasil maka akan dikembalikan response:

Latihan 2

Pada latihan ini akan dibuat sebuah service producer yang akan mengembalikan daftar airport yang ada di suatu kota di Indonesia. API yang digunakan pada latihan ini merupakan API yang dapat membantu pencarian *airport* menggunakan autocomplete dengan term parameter.

Misalnya kita ingin mencari *airport* di Jakarta, kita bisa menggunakan keyword “Jak” atau keyword lain yang merupakan awalan dari jakarta, API akan mengembalikan response berupa value dan label.
→ **value:** tiga huruf IATA location code dari *airport* dan kota.
→ **label:** nama dari airport + IATA location code yang dituliskan dalam [...]

1. Pada class *Setting* masukan url API beserta API key, serta masukan query “country = ID” untuk mem-*filter* kota yang ada di indonesia.

2. Instansiasi sebuah *object* **RestTemplate** dengan menggunakan anotasi **@Bean**

Anotasi **@Bean** digunakan untuk melakukan proses instansiasi object pada spring

RestTemplate digunakan untuk melakukan simplifikasi terhadap pemanggilan HTTP request dari client. RestTemplate menyediakan *method* untuk membuat *HTTP request* dan *handling response*

Setelah instansiasi, inject dependency dengan menggunakan anotasi **@Autowired**

3. Untuk melakukan *request GET* implementasikan method getAirportDetail

```
@GetMapping(value= "/view")
public String getAirportDetail(@RequestParam(value = "term") String term) throws Exception {
    String path = Setting.flightUrl + "&term=" + term;
    return restTemplate.getForEntity(path, String.class).getBody();
}
```

- Anotasi **@GetMapping** merupakan shortcut dari anotasi **@RequestMapping(method = RequestMethod.GET)**

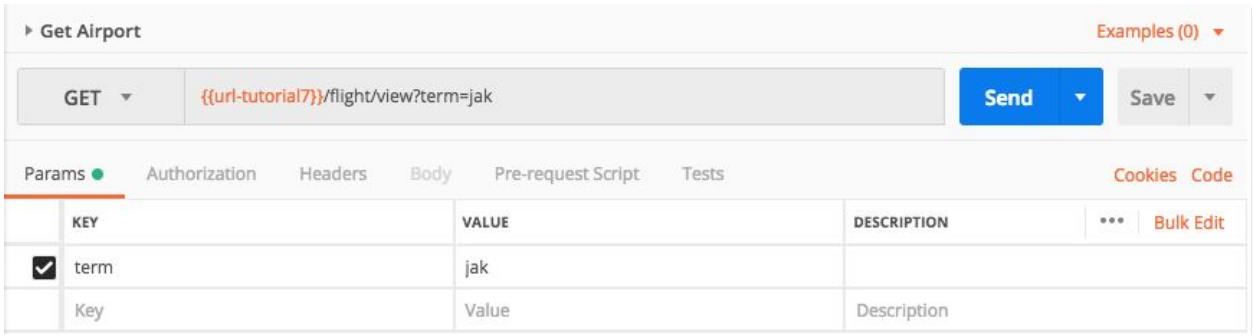
- Method **getAirportDetail()** akan menerima parameter term yaitu awalan huruf dari suatu kota misalnya “Jak” untuk “Jakarta”

- Ambil url dari Setting dengan menambahkan parameter term

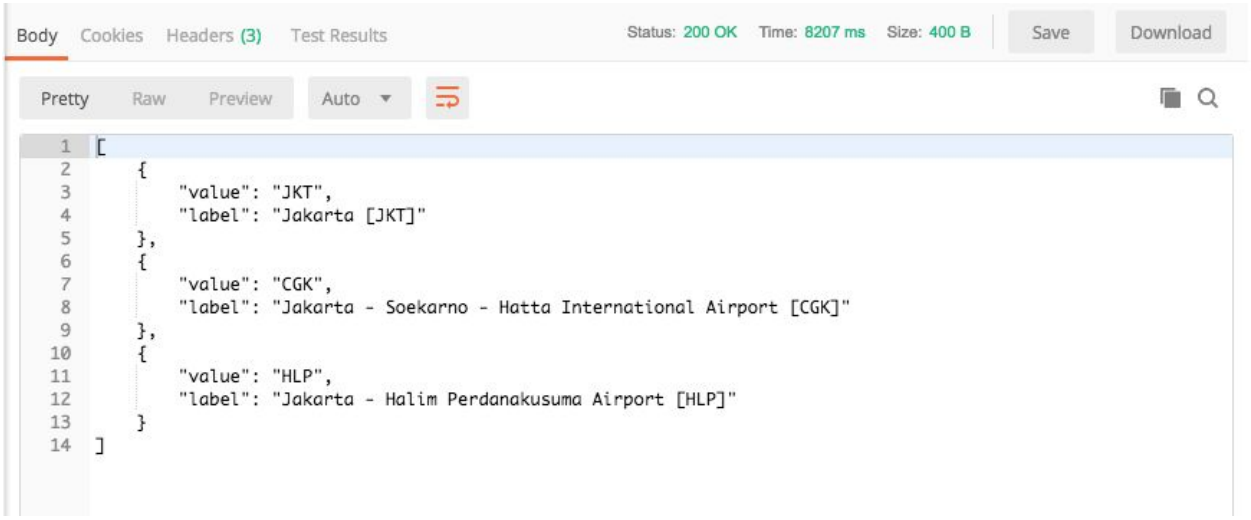
- Untuk mengembalikan response berupa raw json gunakan method dari **RestTemplate** yaitu **getForEntity,** getForEntity akan mengembalikan error 404 ketika data tidak ditemukan tidak seperti **getForObject** yang akan mengembalikan status code 200 jika object null.

  → Parameter1: **path** merupakan url dari API
  → Parameter 2: **String.class** merupakan kelas dari string karena parameter yang dibutuhkan adalah kelas bukan object String
  → method **.getBody()** mengembalikan response body.

4. Testing pada postman
   Masukan request baru *Get Airport,* dan akses url sebagai berikut:



Send request, maka akan dikembalikan *response airport* yang berada di jakarta dan 3 huruf IATA *location code* yang berhubungan dengan jakarta



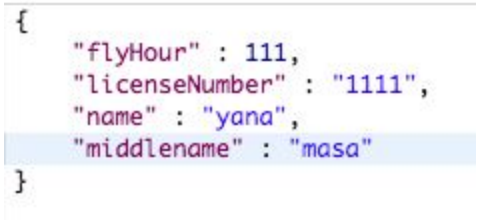
Soal Write-up

//di soal lab ga ada tapi kata kak achim pake :((

1. Kegunaan JsonIgnoreProperties?

@JsonIgnoreProperties digunakan untuk seralization dan deserialization dari JSON ke java object. Anotasi ini akan mengabaikan *properties* yang ada pada JSON namun tidak ada pada kelas.

Misalnya jika JSON input pada pilot



sementara pada kelas pilot hanya terdapat attribute ***flyHour, licenseNumber,*** dan ***name,*** maka **middleName** akan diabaikan dan tidak akan terjadi error karena *deserialization* akan melakukan *throwing exception*

2. Jika tidak ada handler untuk relasi antar model akan terjadi error, jelaskan!

3. Perbedaan RestController ama Controller?

- **@Controller** hanya menandakan sebuah kelas sebagai Spring MVC controller. Anotasi ini tidak dapat digunakan untuk menampilkan response berupa JSON atau XML, jika ingin menampilkan response JSON atau XML harus ditambah penggunaan **@ResponseBody**

- **@RestController** adalah versi anotasi controller yang lebih praktis karena merupakan gabungan dari **@Controller** dan **@ResponseBody** sehingga dapat menggembalikan response JSON atau XML. Penggunaan template engine seperti thymeleaf tidak bisa menggunakan **@RestController.**

4. Perbedaan penggunaan RequestMapping(Methodtype.GET) dengan GetMapping?

@GetMapping merupakan shortcut dari anotasi **@RequestMapping(method = RequestMethod.GET)**

5. Mengapa beberapa producer mengharuskan penggunaan API-key?

API key digunakan untuk mengidentifikasi project, app, atau site yang memanggil API tersebut. API key dapat melakukan beberapa hal sebagai berikut:

- Block *anonymous traffic*
- Mengontrol jumlah pemanggilan API
- Mengidentifikasi pola penggunaan API
- Melakukan filter log berdasarkan API key

6. Perbedaan JsonIgnore dan JsonIgnoreProperties.

Kedua fungsi anotasi jackson ini memiliki fungsi yang sama yaitu mengabaikan *properties* yang terdapat pada JSON namun tidak terdapat pada kelas pada proses *serialization* dan *deserialization.*
@JsonIgnore merupakan anotasi yang berada pada *method-level*

@JsonIgnoreProperties merupakan anotasi yang berada pada class-level

7. Jelaskan apa yang bisa menjadi penyebab error 403 pada pemanggilan API?

HTTP 403 terjadi saat server mengerti permintaan client namun menolak untuk memberi response karena beberapa alasan:

- 403.4 - SSL required
- 403.7 - Client certificate required.
- 403.8 - Site access denied.
- 403.9 - Too many users.
- 403.10 - Invalid configuration.
- 403.11 - Password change.
- 403.12 - Mapper denied access.
- 403.13 - Client certificate revoked.
- 403.14 - Directory listing denied.
- 403.15 - Client Access Licenses exceeded.
- 403.16 - Client certificate is untrusted or invalid.
- 403.17 - Client certificate has expired or is not yet valid.
- 403.18 - Cannot execute request from that application pool.
- 403.20 - Passport logon failed.
- 403.21 - Source access denied.
- 403.502 - Too many requests from the same client IP; Dynamic IP Restriction limit reached.