

LATIHAN

1. Pada Latihan ke 1, saya mempelajari bagaimana cara membuat web service, akan tetapi menggunakan database sendiri. Disini, tools yang digunakan adalah Postman untuk menguji REST API yang sudah kita buat.

Hal-hal yang sudah saya lakukan :

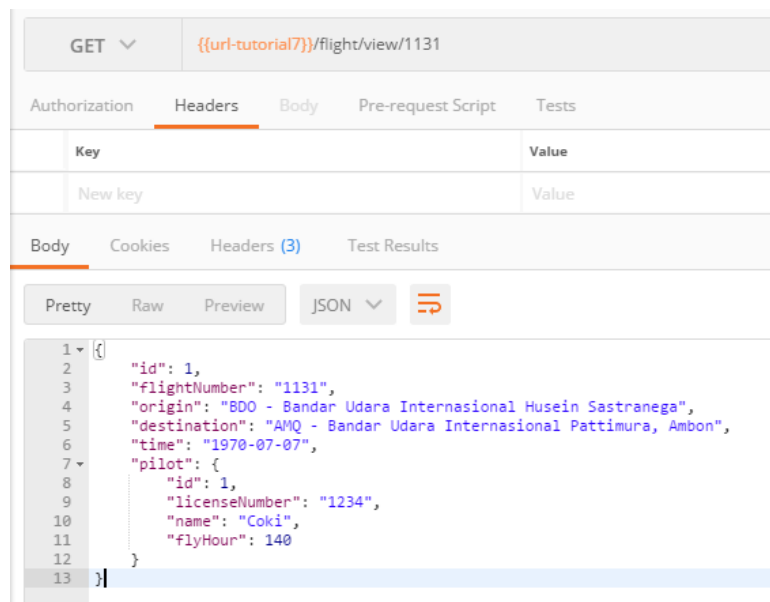
1. Setting environment dan Collection pada API – ini bertujuan untuk memudahkan kita dalam membedakan environment untuk API Dev dan API yang sedang live.
2. Selanjutnya, untuk melakukan testing tiap method, maka kita mengujinya satu persatu menggunakan env base_url (dalam tutorial ini yaitu {{url-tutorial7}}. Kegunaannya adalah agar kita bisa mengetahui apakah hasil yang degenerate dari source code sudah sesuai ekspektasi/belum. Disini, untuk bagian add-update-dsb kita menggunakan JSON, json sendiri adalah suatu metode pertukaran data/informasi berupa teks.
3. Kemudian, hasil test tersebut dapat kita simpan sesuai collection mana.

FlightController setelah *adjust*

Url	Source Code
POST add flight {url}/flight/add	<pre>@PostMapping(value = "/add") private FlightModel addFlightSubmit(@RequestBody FlightModel flight) { return flightService.addFlight(flight); }</pre>
DELETE flight {url}/flight/{flightID}	<pre>@DeleteMapping(value = "{flightId}") private String deleteFlight(@PathVariable("flightId") long id) { FlightModel flight = flightService.getFlightDetailById(id); flightService.deleteById(id); return "flight has been deleted" ; }</pre>
PUT update flight {url}/flight/update/{flightID}? destination={destination}&origin={ orig in}&time={time} parameter dapat tidak harus lengkap	<pre>@PutMapping(value = "/update/{flightId}") private String updateFlightSubmit(@PathVariable (value="flightId") long flightId, @RequestParam("destination") String destination, @RequestParam("origin") String origin, @RequestParam("time") Date time) { FlightModel flight = flightService.getFlightDetailById(flightId); if(flight.equals(null)) { return "Couldn't find any flight details"; } flight.setDestination(destination); flight.setOrigin(origin); flight.setTime(time); flightService.updateFlight(flightId, flight); }</pre>

	<pre> return "flight update success"; } }</pre>
GET flight {url}/flight/view/{flightNumber}	<pre>@GetMapping(value = "/view/{flightNumber}") private FlightModel viewFlight(@PathVariable ("flightNumber") String flightNumber){ FlightModel flight = flightService.getFlightDetailByFlightNumber(flightNum ber); return flight; } }</pre>
GET all flight {url}/flight/all	<pre>@GetMapping(value = "/all") private List<FlightModel> viewAllFlight(){ List<FlightModel> listFlight = flightService.getListOfFlight(); for(FlightModel flight : listFlight) { flight.setPilot(null); } return listFlight; } }</pre>

#TampilanViewAFlight



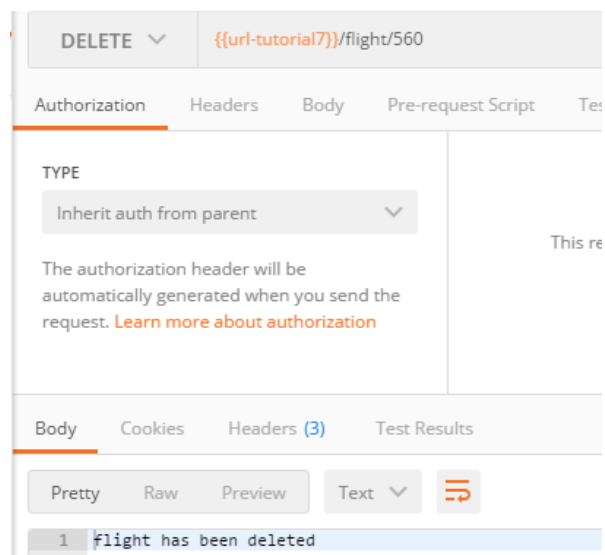
#TampilanViewAllFlight

```
GET {{url-tutorial7}}/flight/all

Pretty Raw Preview JSON
```

```
4453   "origin": "PDG - Bandar Udara Internasional Minangkabau, Pada",
4454   "destination": "UPG - Bandar Udara Internasional Sultan Hasanuddin",
4455   "time": "1980-09-19",
4456   "pilot": null
4457 },
4458 {
4459   "id": 558,
4460   "flightNumber": "1592",
4461   "origin": "AMQ - Bandar Udara Internasional Pattimura, Ambon",
4462   "destination": "UPG - Bandar Udara Internasional Sultan Hasanuddin",
4463   "time": "1980-10-25",
4464   "pilot": null
4465 },
4466 {
4467   "id": 559,
4468   "flightNumber": "1639",
4469   "origin": "KOE - Bandar Udara Internasional El Tari, Kupang",
4470   "destination": "UPG - Bandar Udara Internasional Sultan Hasanuddin",
4471   "time": "1992-02-01",
4472   "pilot": null
4473 },
4474 {
4475   "id": 560,
4476   "flightNumber": "1651",
4477   "origin": "TNJ - Bandar Udara Internasional Raja Haji Fisabil",
4478   "destination": "UPG - Bandar Udara Internasional Sultan Hasanuddin",
4479   "time": "1992-02-01",
4480   "pilot": null
4481 }
4482 ]
```

#TampilanDeleteAFlight

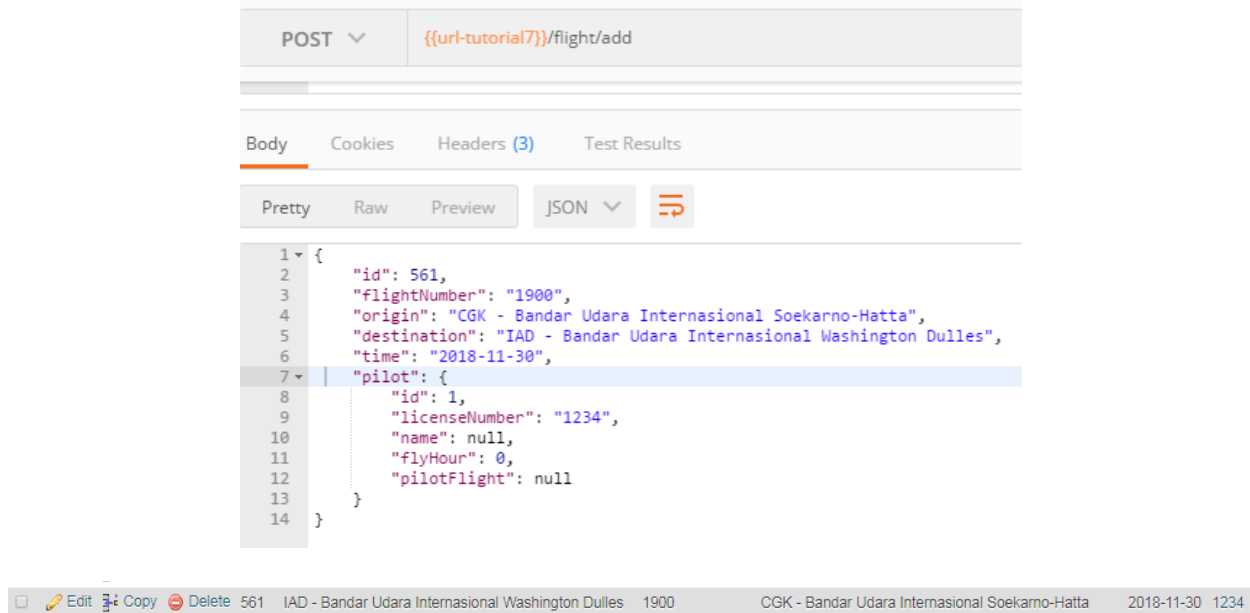


<input type="checkbox"/>				557	UPG - Bandar Udara Internasional Sultan Hasanuddin	1580	PDG - Bandar Udara Internasional Minangkabau, Pada	1980-09-19	1234
<input type="checkbox"/>				558	UPG - Bandar Udara Internasional Sultan Hasanuddin	1592	AMQ - Bandar Udara Internasional Pattimura, Ambon	1980-10-25	1234
<input type="checkbox"/>				559	UPG - Bandar Udara Internasional Sultan Hasanuddin	1639	KOE - Bandar Udara Internasional El Tari, Kupang	1992-02-01	1234

↑ ☐ Check all With selected:

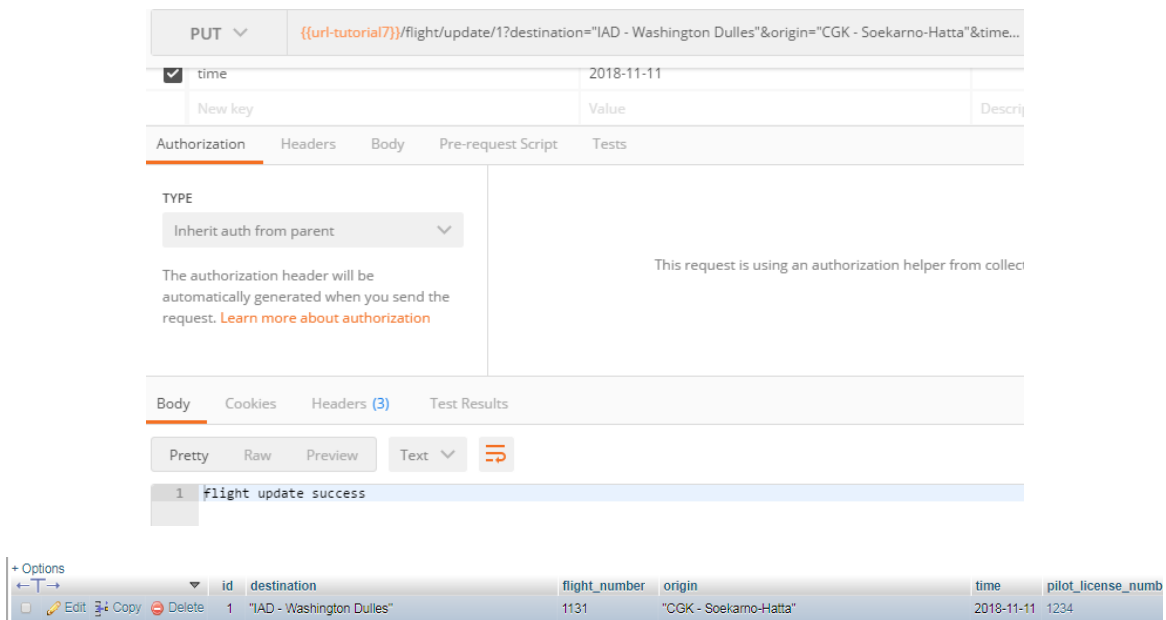
^ hasil delete, di db tidak ditemukan lagi flight dengan Id 560

#TampilanAddAFlight

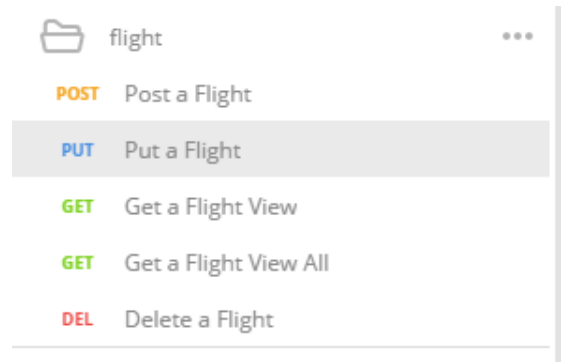


^hasil add di Database

#TampilanUpdateAFlight



^hasil update flight dengan ID 1 di database



^Struktur akhir folder flight

2. Pada latihan kedua, saya mempelajari mengakses web service hanya saja dengan menggunakan API milik orang lain, dalam hal ini yaitu :

<https://sandbox.amadeus.com/hybridauth/endpoint?hauth.done=GitHub&code=0ca72104cd425b683c9b>

Dengan mengakses API ini, saya bisa mendapatkan informasi terkait airports dari database mereka, tanpa perlu mengetahui bagaimana cara mereka menyimpan data tersebut dalam databasenya. Informasi yang digenerate berupa Json.

Langkah-langkahnya sebagai berikut :


1. Mula-mula saya membuat ModelController yang berperan mengatur REST API milik sandbox yang akan saya akses nantinya. Disini, karena saya akan mengambil data dari mereka, maka yang digunakan adalah method “GET”—yang ditunjukkan melalui anotasi @GetMapping, dengan value mapping url menyesuaikan controller saya yaitu model/airports/{city}. Kenapa ada data city? Karena pada website sandbox menjelaskan bahwa untuk API tersebut, parameter yang harus dipenuhi berupa “city”.
2. Berikutnya, saya membuat path yang akan melakukan mapping url yang diakses dengan API mereka dan menambahkan parameter yang nantinya akan dibutuhkan. Method ini akan mengembalikan rest template yang mengambil entity/informasi dari API sandbox.
3. Pada step ke 2, untuk mengakses API Sandbox, saya memerlukan url API mereka. Untuk mendapatkan url ini, saya harus terlebih dahulu login dan mendaftarkan aplikasi saya di website mereka. Nantinya, akan digenerate sebuah API Key yang unik, yang bisa saya gunakan untuk mendapatkan url yang dibutuhkan.
4. Url yang saya dapat tersebut kemudian saya masukkan/update sebagai flightUrl static di Settings. Java, sehingga setiap mengakses API tersebut, bisa menggunakan url ini.
5. Setelah semuanya selesai disetting, baru dilakukan testing dengan menggunakan postman yang menguji url tsb. Hasilnya, dikembalikan informasi berupa informasi data airport yang terdapat di Jakarta. Jakarta adalah kota yang saya masukkan sebagai parameter pencarian.

GET `{{url-tutorial7}}/model/airports/Jakarta`

Authorization Headers Body Pre-request Script Tests

Key	Value
New key	Value

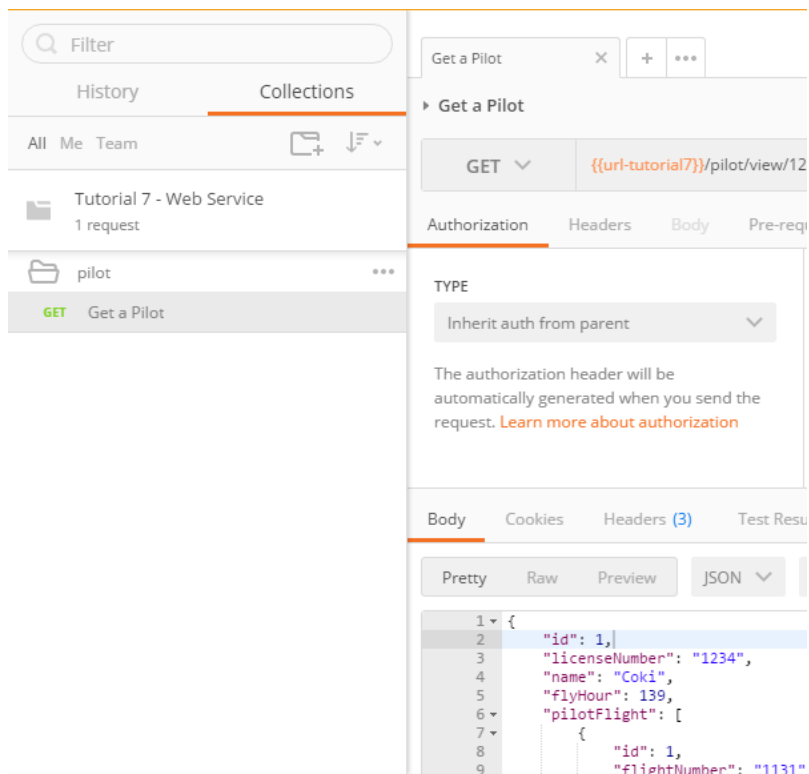
Body Cookies Headers (3) Test Results

Pretty Raw Preview Text 

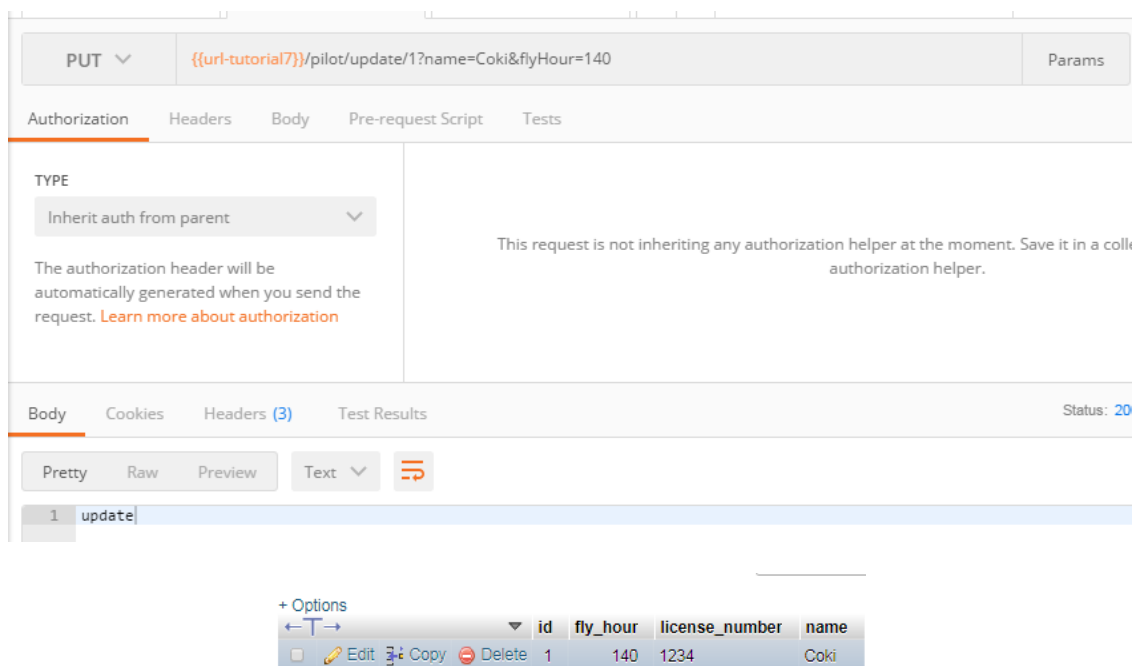
```
1  [
2    {
3      "value": "JKT",
4      "label": "Jakarta [JKT]"
5    },
6    {
7      "value": "CGK",
8      "label": "Jakarta - Soekarno - Hatta International Airport [CGK]"
9    },
10   {
11     "value": "HLP",
12     "label": "Jakarta - Halim Perdanakusuma Airport [HLP]"
13   }
14 ]
```

Tutorial

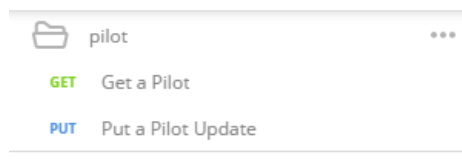
#Tampilan Get a Pilot



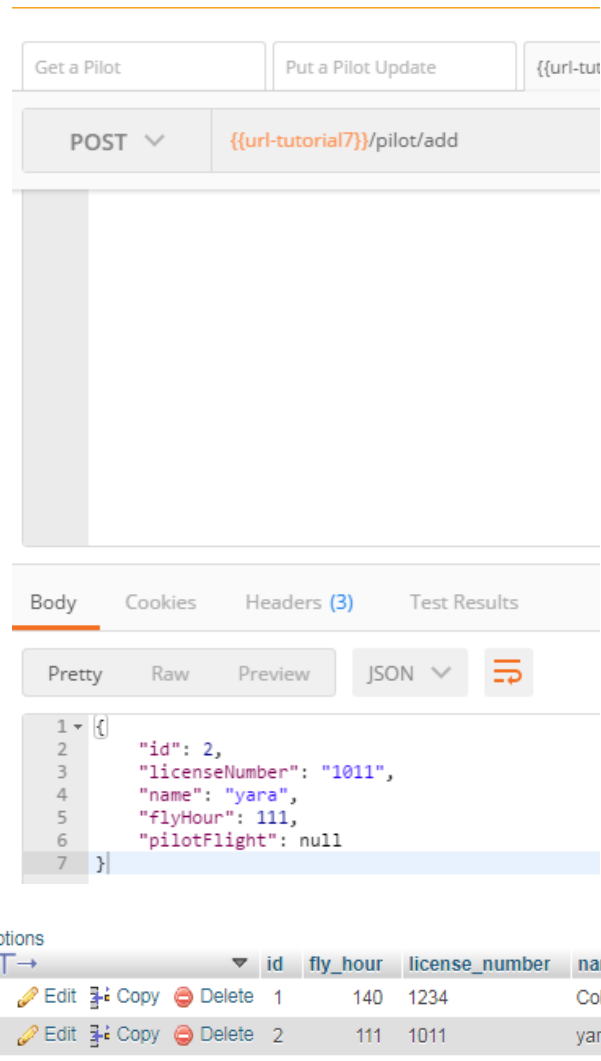
#TampilanPutAPilot (update)



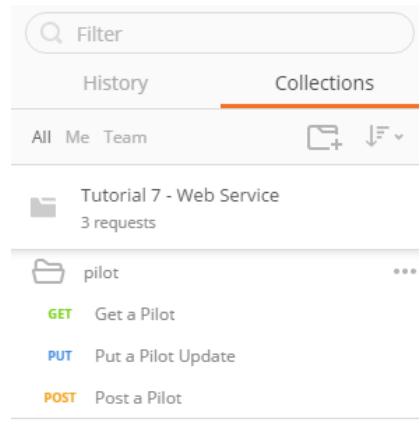
^ terupdate di Database



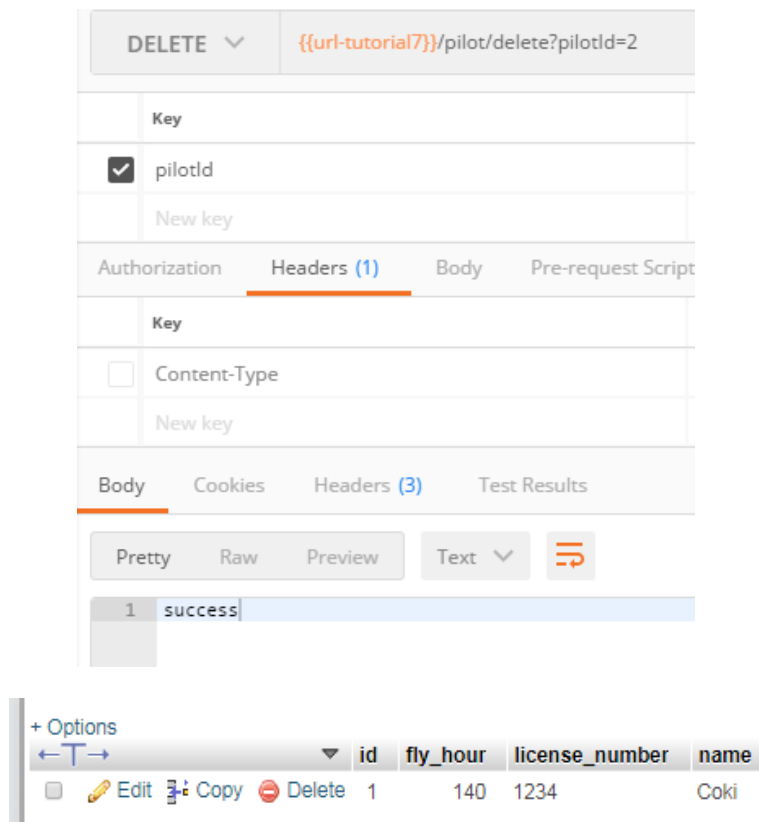
#TampilanPostAPilot



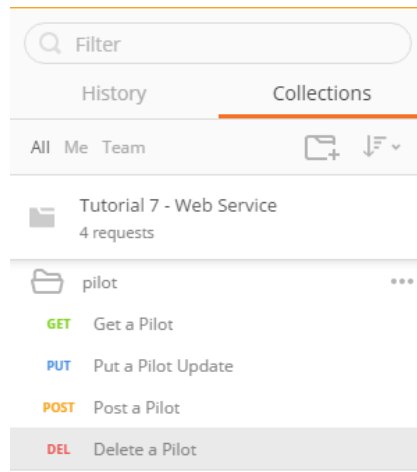
^ hasil menambahkan pilot



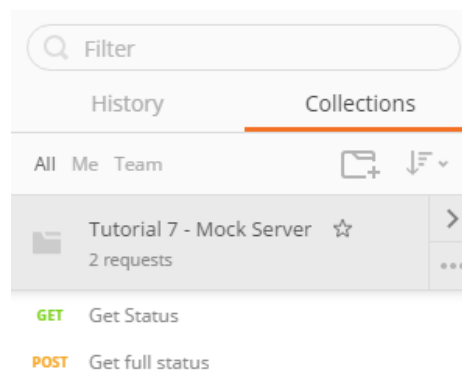
#TampilanDeleteAPilot



^ hasil setelah pilot dengan id 2 di delete, terupdate di Db



MOCK SERVER

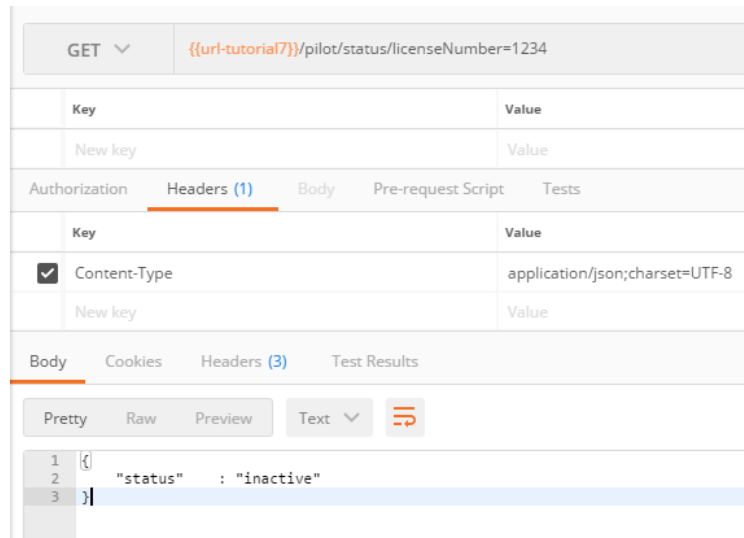


Tutorial 7 - Mock Server mock server created

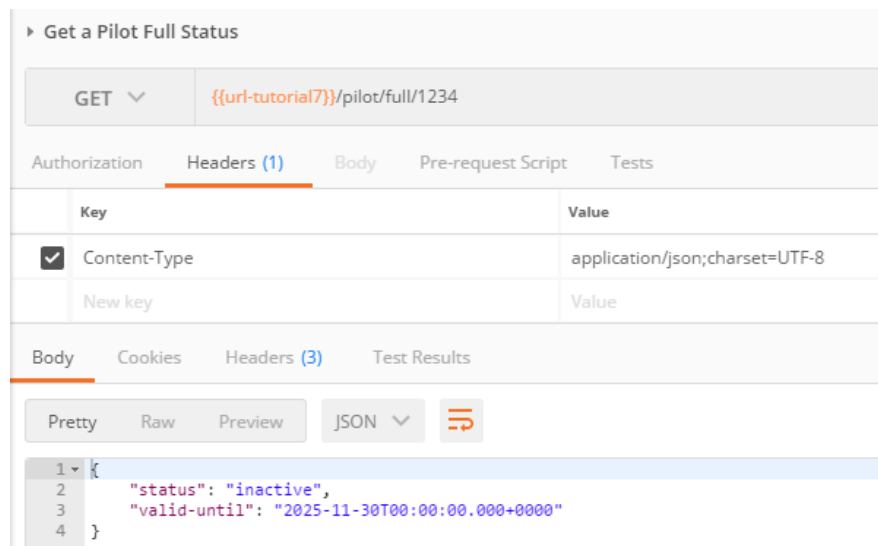
A mock server responds with sample responses, without an actual backend

NEXT STEPS

- **Send a request to this mock URL**
<https://4ae82f75-1905-4c4a-85df-94f75b8357c0.mock.pstmn.io/> followed by the request path
- **Modify the response of the mock server**
Editing the **Example** of the request in the collection would change the response served by the mock server. [Learn more](#)



^ Hasil mockserver get status



^ hasil mock server get full status

Kesimpulan :

Web service adalah aplikasi sekumpulan data (*database*), perangkat lunak (*software*) atau bagian dari perangkat lunak yang dapat diakses secara remote dengan sebuah perantara, dalam tutorial ini memakai postman, API sandbox, dan Springboot. Secara umum, *web service* dapat diidentifikasi dengan menggunakan URL seperti hanya web pada umumnya. Namun yang membedakan *web service* dengan web pada umumnya adalah interaksi yang diberikan oleh *web service*. Berbeda dengan URL *web* pada umumnya, URL *web service* hanya mengandung kumpulan informasi, perintah, konfigurasi atau sintaks yang berguna membangun sebuah fungsi-fungsi tertentu dari aplikasi (misal pada tutorial ini : Json). Web

WriteUp

Tutorial 7 – Faizah Afifah – 1606918093 – APAP C

service juga bisa dikatakan suatu metode pertukaran data, tanpa perlu memperhatikan bagaimana suatu data disimpan dalam database tsb, dengan Bahasa apa, dsb.