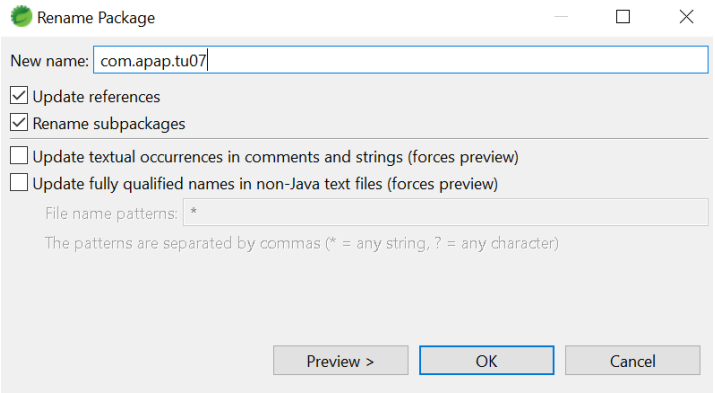
Tutorial 07 – Web Service

Tutorial ini mengasumsikan penggunaan pada *Windows*, tanpa melarang penggunaan OS lain, silakan menyesuaikan.

1. **Pendahuluan**
   - Install Postman, unduh pada link berikut: https://www.getpostman.com/downloads/
   - Buat akun Postman, dapat dilakukan pada *website* atau aplikasi.
   - Jalankan aplikasi Postman, lalu login menggunakan akun yang sudah dibuat.
   - Unduh Maven Project (Tutorial-06) yang ada di Scele lalu ekstrak (tip: pada *workspace*). Agar berbeda dengan *project* sebelumnya ubah nama folder menjadi tutorial-07.
   - Jalankan STS, lalu *import project* tersebut.
   - Ubah nama package dari *.tu06 menjadi *.tu07, klik kanan package >> Refactor >> Rename



   - Buat DB dengan nama tu07, lalu import SQL yang ada di Scele (Tutorial-06 schema dan data).
   - Sesuaikan **application.properties** dengan *environment* Anda. Tambahkan custom port:

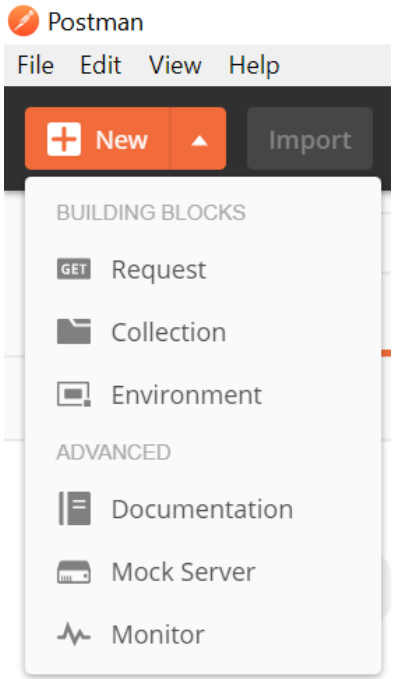
```
# Custom port
server.port = 2016
```

2. **Membuat RestController**
   - Ubah PilotController menjadi sbb.:

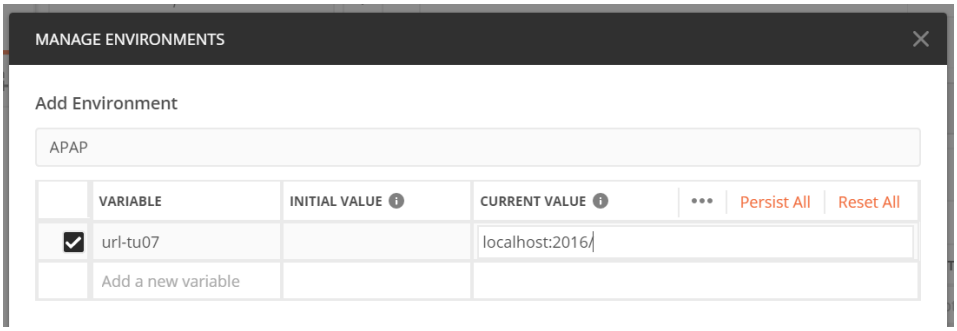
```
package com.apap.tu07.controller;

import com.apap.tu07.model.PilotModel;
import com.apap.tu07.service.PilotService;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;
```

```java
@RestController
@RequestMapping("/pilot")
public class PilotController {
    @Autowired
    private PilotService pilotService;

    @PostMapping(value = "/add")
    public PilotModel addPilotSubmit(@RequestBody PilotModel pilot) {
        return pilotService.addPilot(pilot);
    }

    @GetMapping(value = "/view/{licenseNumber}")
    public PilotModel pilotView(@PathVariable("licenseNumber") String licenseNumber) {
        PilotModel pilot = pilotService.getPilotDetailByLicenseNumber(licenseNumber).get();
        return pilot;
    }

    @DeleteMapping(value = "/delete")
    public String deletePilot(@RequestParam("pilotId") long pilotId) {
        PilotModel pilot = pilotService.getPilotDetailById(pilotId).get();
        pilotService.deletePilot(pilot); // Buat implementasi method ini
        return "success";
    }

    @PutMapping(value = "/update/{pilotId}")
    public String updatePilotSubmit(@PathVariable("pilotId") long pilotId,
            @RequestParam("name") String name,
            @RequestParam("flyHour") int flyHour) {
        PilotModel pilot = pilotService.getPilotDetailById(pilotId).get();
        if (pilot.equals(null)) {
            return "Couldn't find your pilot";
        }

        pilot.setName(name);
        pilot.setFlyHour(flyHour);
        pilotService.updatePilot(pilotId, pilot); // Buat implementasi method ini
        return "update";
    }
```

3. **Mengakses method GET**
   - Buka Postman, Klik panah disamping New >> Environment

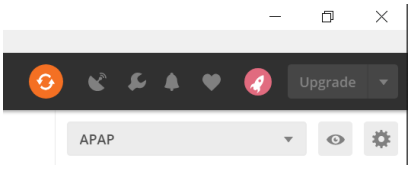


- Buat *environment* baru bernama “APAP” dan buat *variable* dengan spesifikasi sbb.:

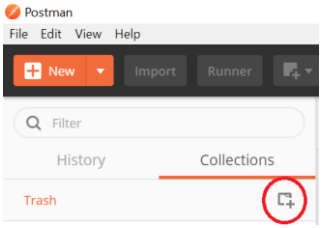


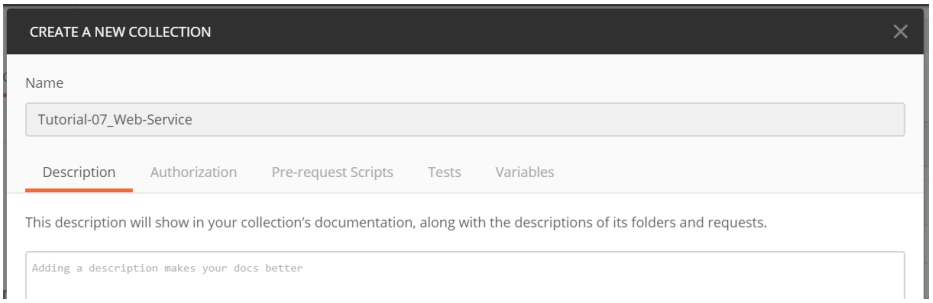
Klik Add, lalu tutup window tersebut.

- Pilih *environment* APAP yang telah dibuat pada bagian kanan atas Postman.

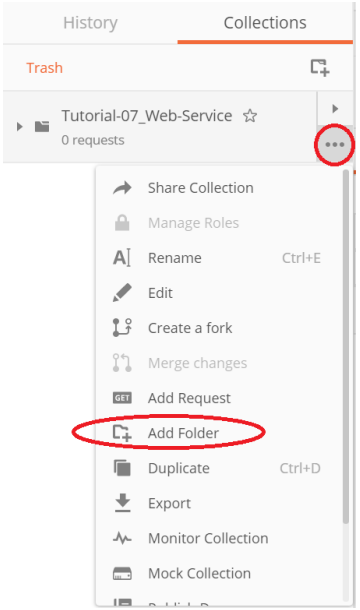


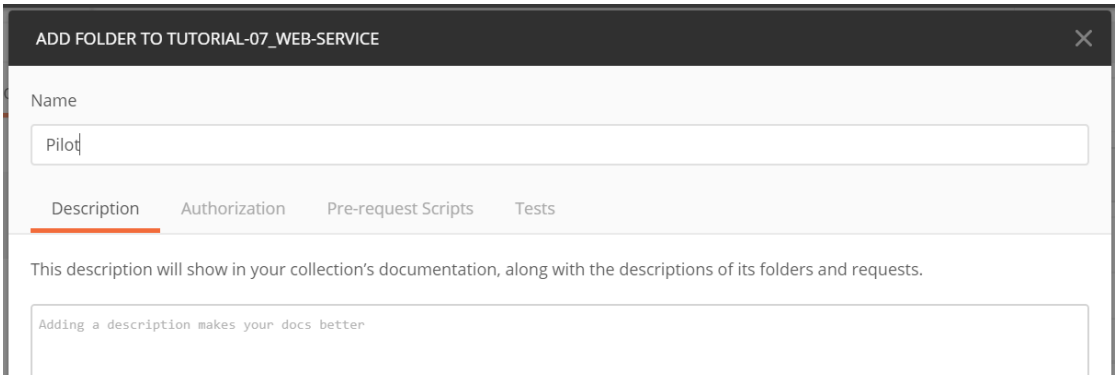
- Buat Collection baru dengan nama “Tutorial-07_Web-Service”





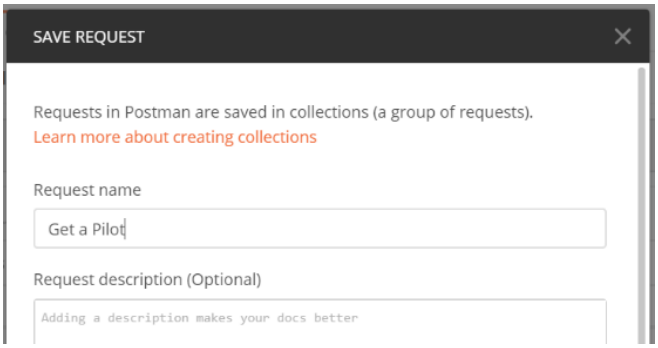
- Buat folder baru dengan mengakses menu dot, beri nama “Pilot”.



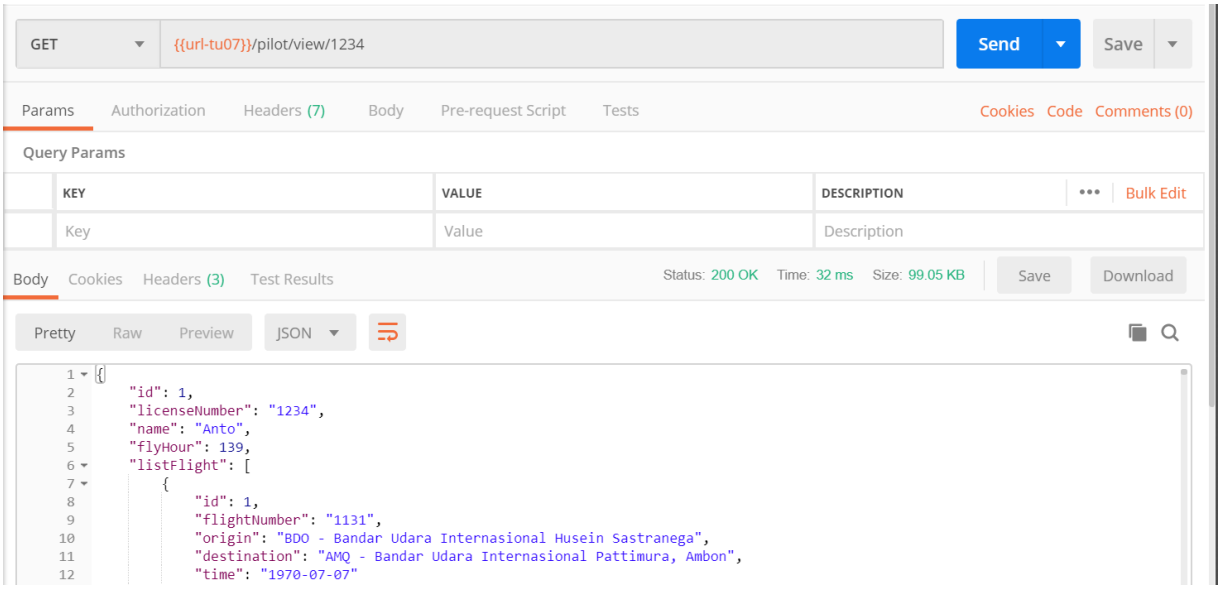


- Buat request baru dengan method GET dengan url "{{url-tu07}}/pilot/view/1234", bilangan 1234 merupakan *license number* pada DB Anda.

- Klik Save >> beri nama "Get a Pilot" dan Save pada folder Pilot yang sudah dibuat.

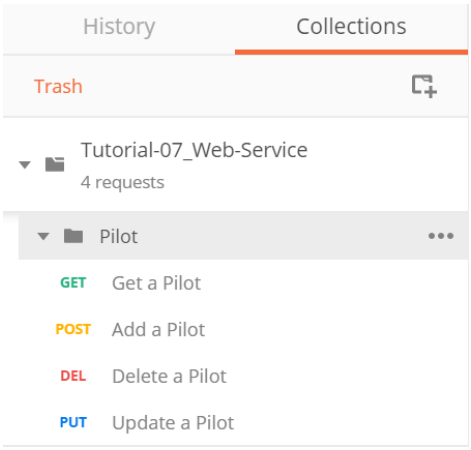


- Kirim *request* dengan menekan tombol "Send". Anda akan menerima pesan pada *response body* di bawah, sbb.:



- Buat request lainnya untuk melengkapi akses ke seluruh *web service* yang ada pada PilotController (empat method). Simpan seluruhnya pada folder Pilot.

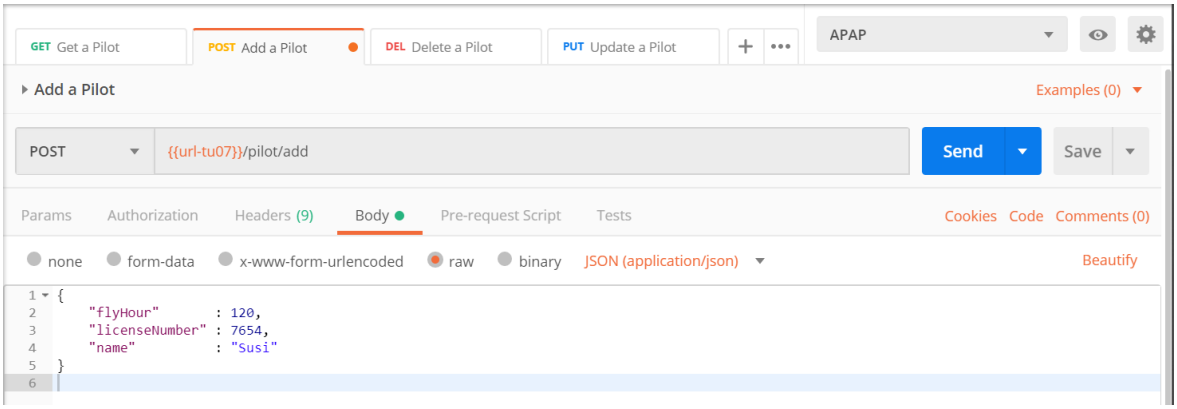
- Hasil akhir pada collection sbb.:



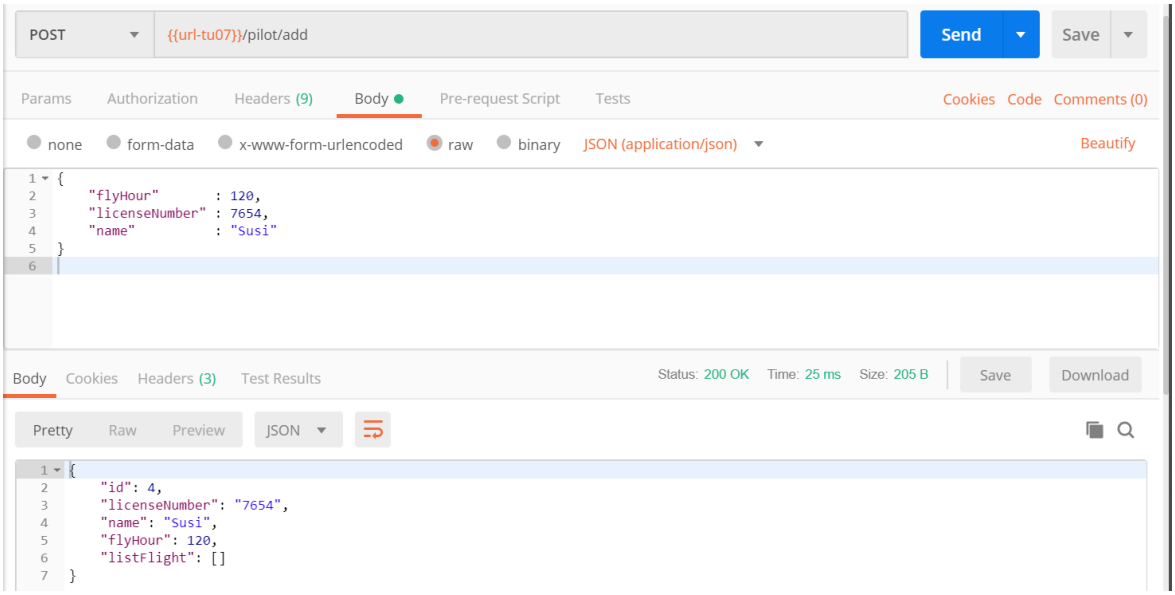
4. Mengakses method POST

Jika menggunakan *field* URL pada browser kita sebenarnya menggunakan *method* GET, sementara *method* lain seperti PUT, DELETE, POST, dll. tidak bisa menggunakan URL browser. Postman digunakan agar bias mengakses tanpa membuat terlebih dahulu sistem yang menerima API dari kita. Terdapat aplikasi lainnya selain Postman.

- Akses *request post* yang telah dibuat untuk "Add a Pilot." Lalu tambahkan *request body* sbb.:

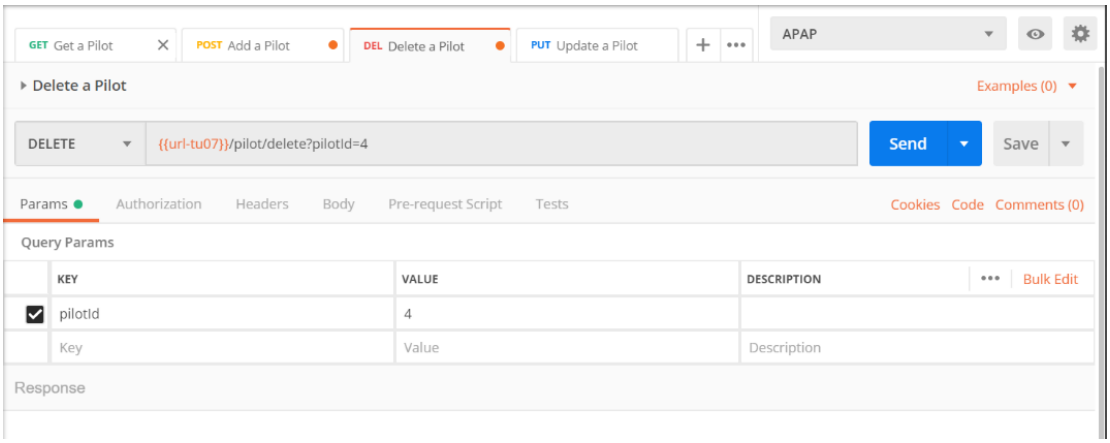


- Kirim *request* dengan menekan tombol "Send". Anda akan menerima pesan pada *response body* di bawah, sbb.:

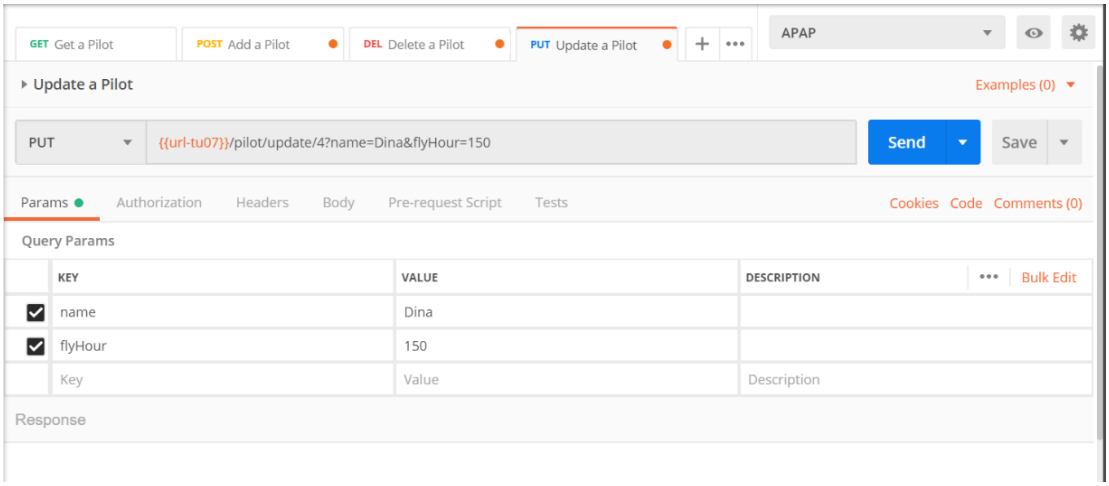


- Cek pada DB apakah data tersebut sesuai.

- Karena pada *delete* dan *update* pilot menggunakan *parameter*, maka jangan lupa untuk menambahkan *paramater* pada Postman.
- Method *delete* misalnya sbb.:



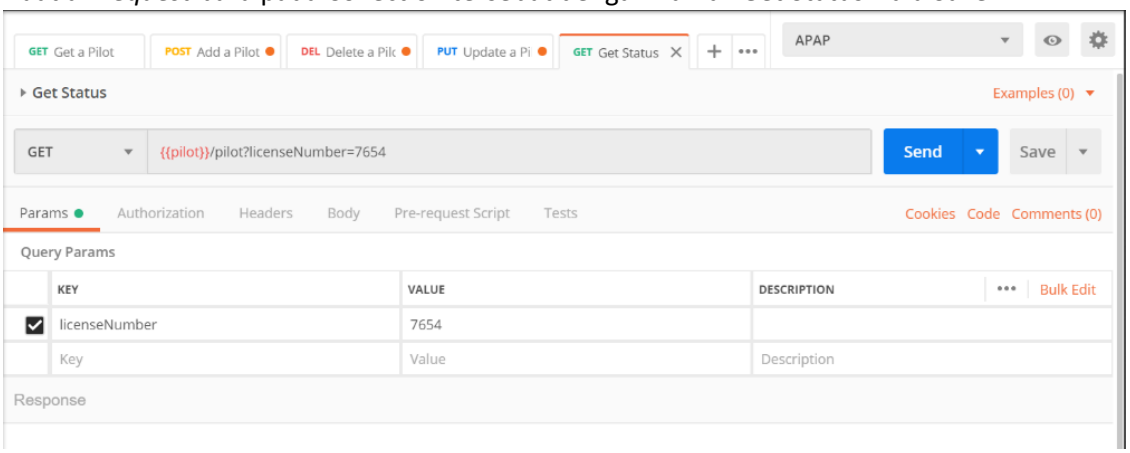
- Method *update* misalnya sbb.:



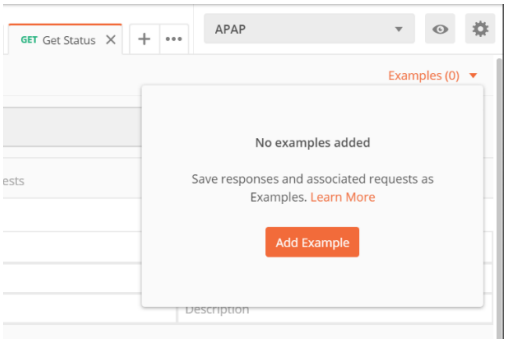
5. Mockserver

Mockserver adalah server bayangan API yang bisa diakses dengan spesifikasi *custom*. Mockserver berguna ketika spesifikasi mengharuskan beberapa sistem untuk berkomunikasi satu sama lainnya sedangkan hanya tersedia API *documentation* dan belum ada implementasinya. Dengan kata lain Mockserver mempermudah percepatan *development* sistem yang sedang dikembangkan secara paralel. Mockserver merupakan *service* yang ada pada Postman.

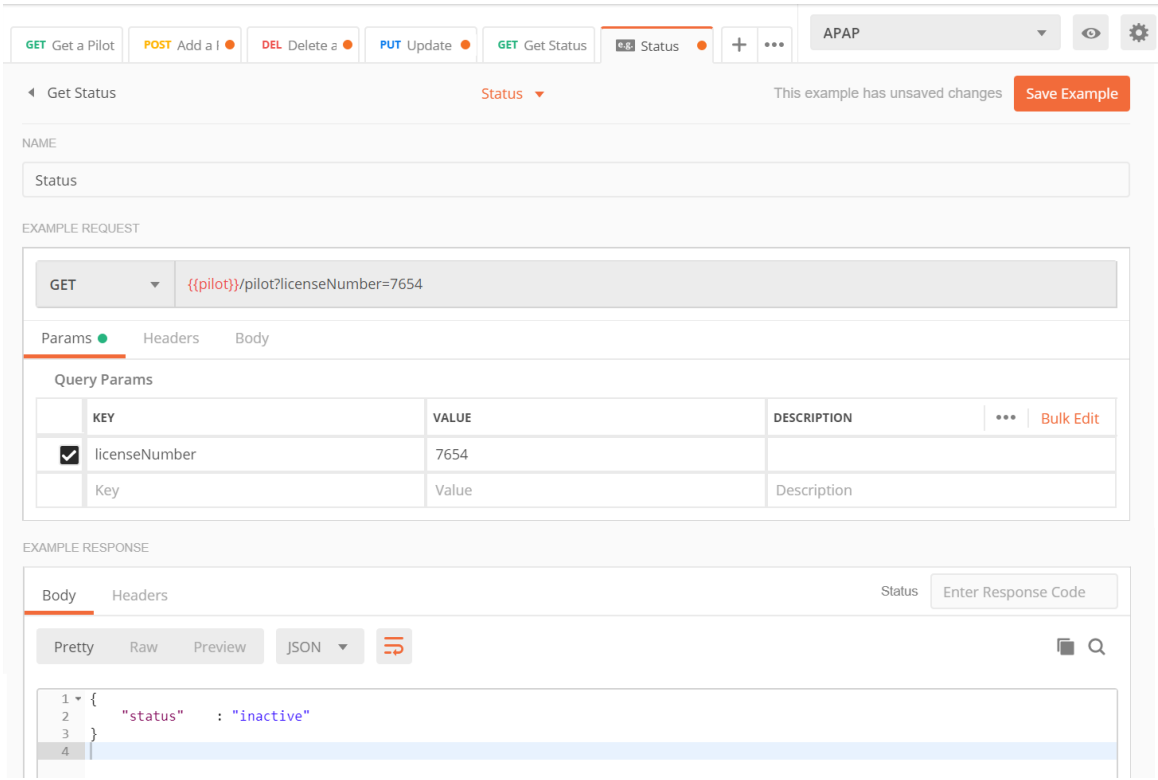
- Buat Collection baru dengan nama “Tutorial-07_Mock-Server”
- Buatlah *request* baru pada Collection tersebut dengan nama “Get Status” lalu Save.



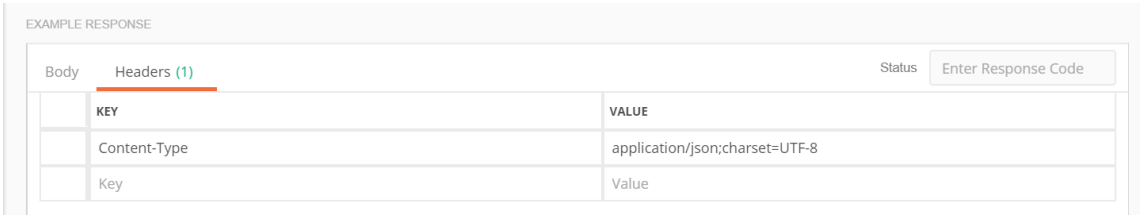
- Klik “Examples (0)” >> “Add Example”



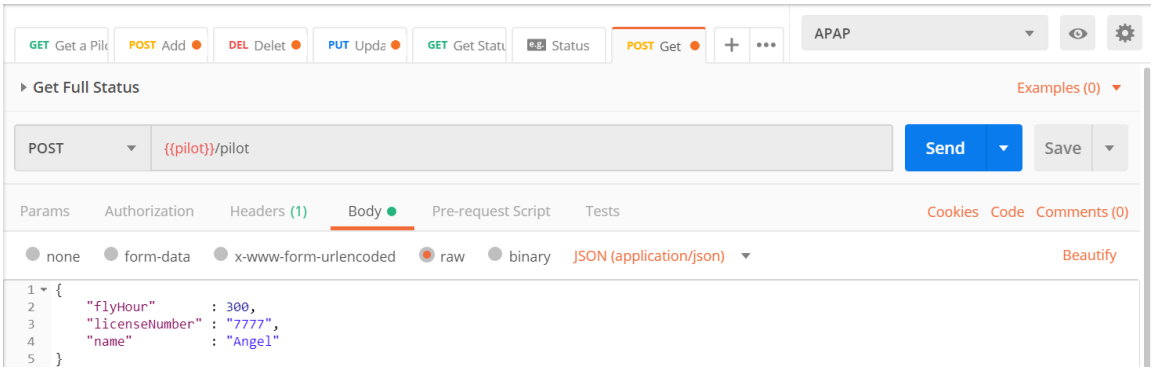
- Buat *example* baru yang bernama “Status” sbb.:



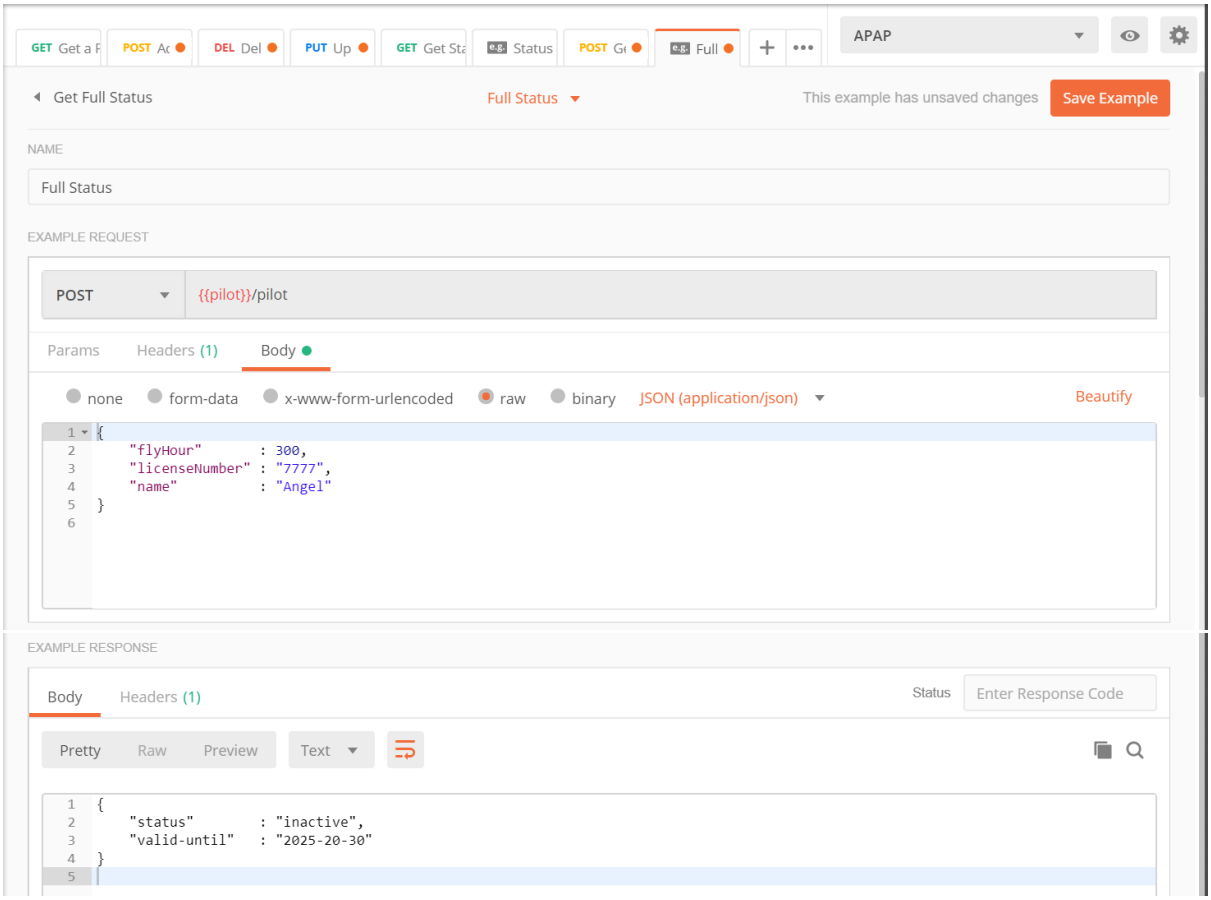
- Untuk *response header* spesifikasinya sbb.:



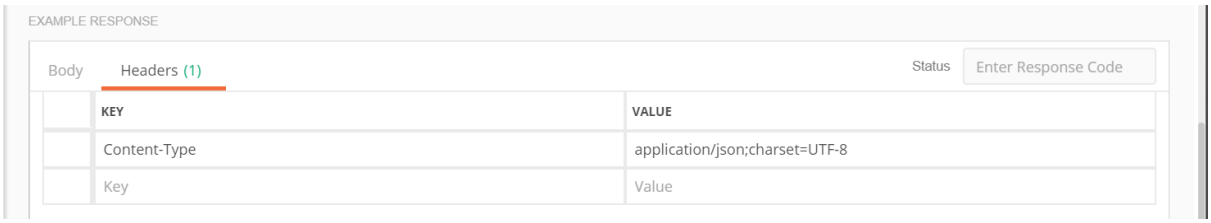
- Buatlah *request* lain dengan nama “Get Full Status” lalu Save.



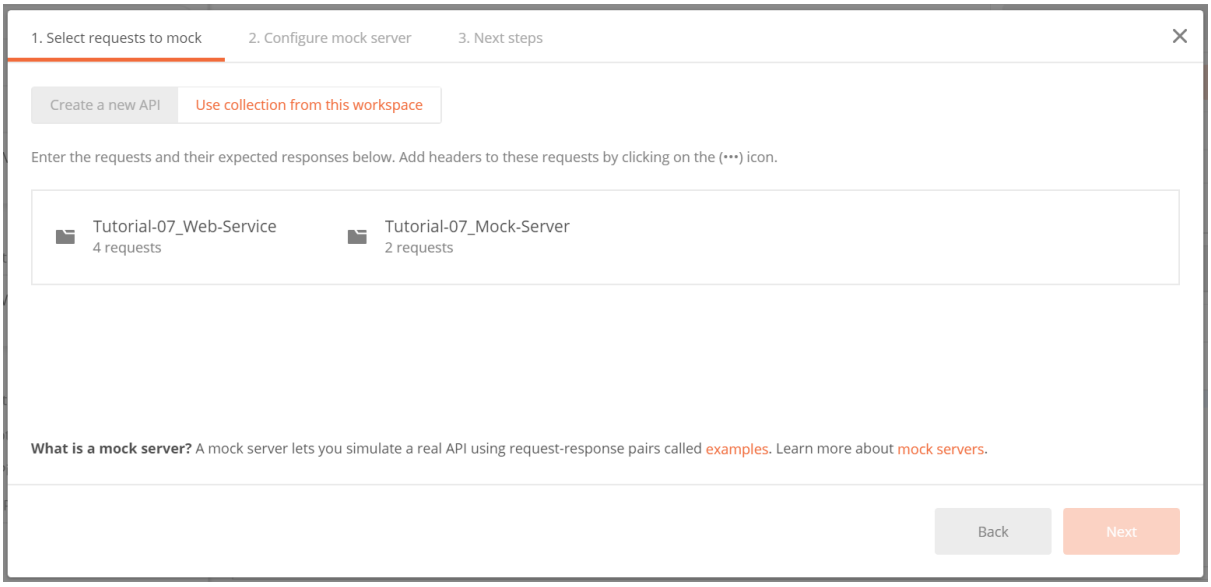
- Buat *example* baru yang bernama “Full Status” sbb.:



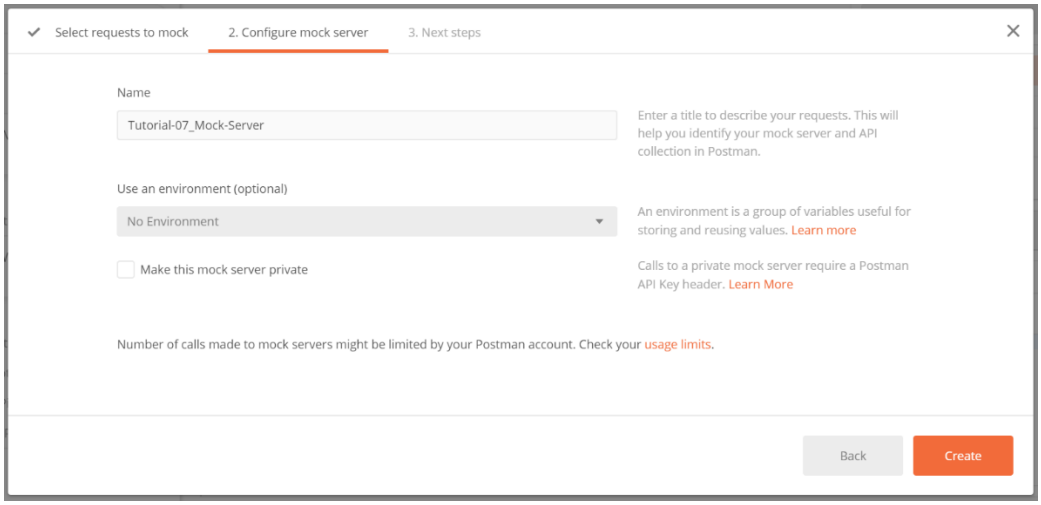
- Untuk *response header* spesifikasinya sbb.:



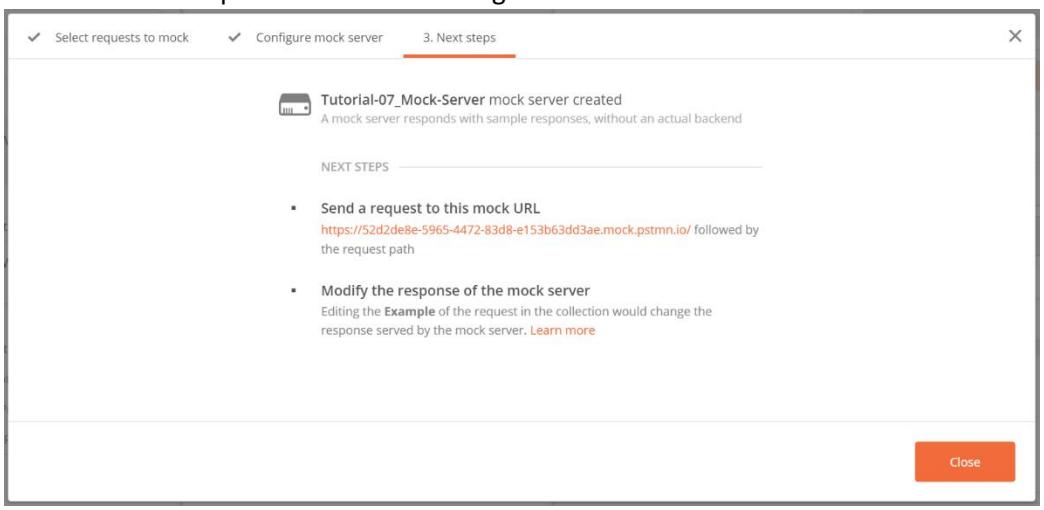
- Klik New >> Mock Server
- Pilih “Use collection from this workspace” >> “Tutorial-07_Mock-Server”



- Klik Create

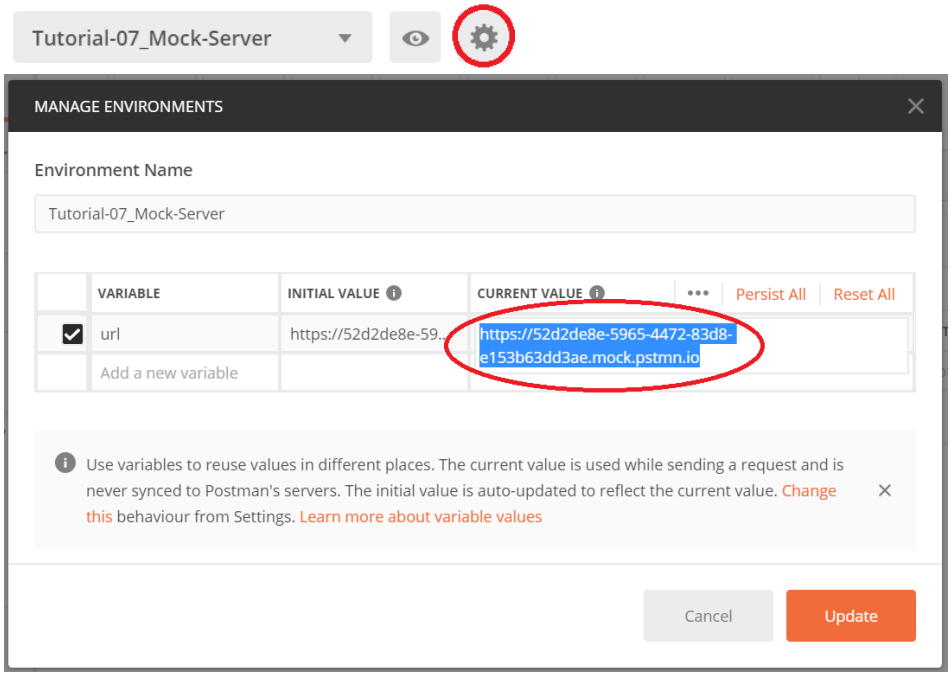


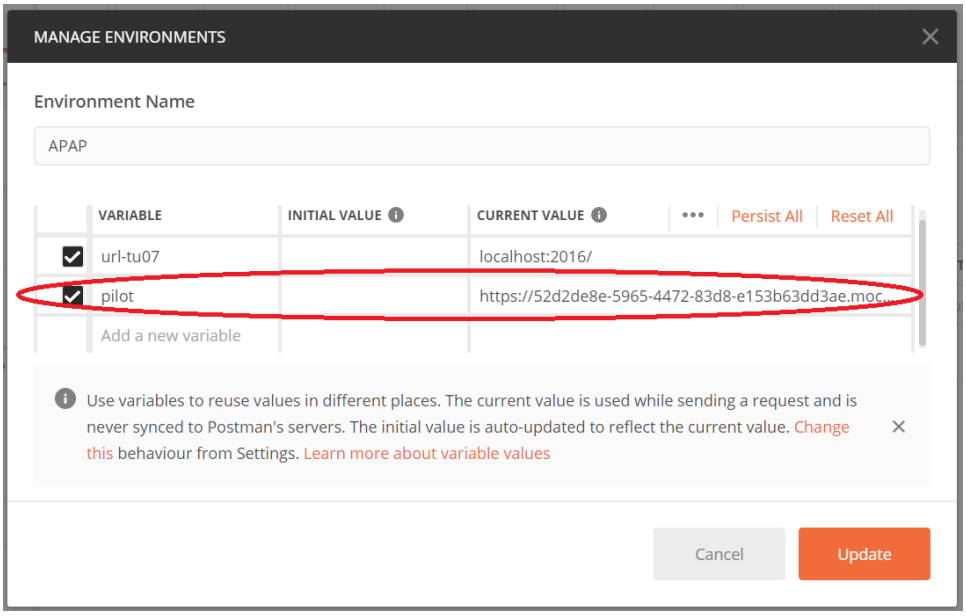
- Anda akan mendapatkan URL untuk mengakses server baru Anda.



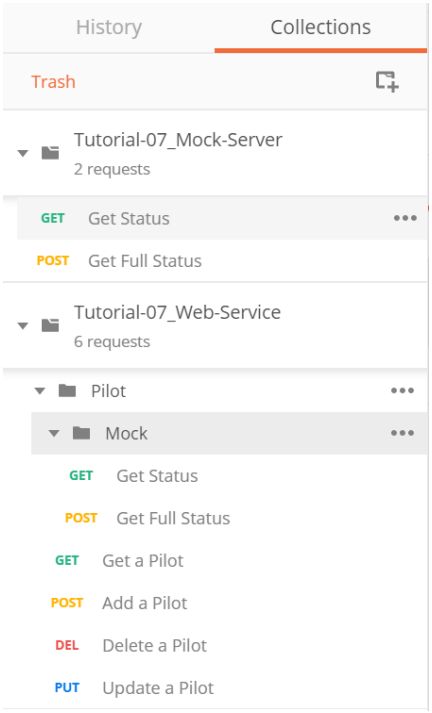
URL tersebut dapat diakses pada *environment* yang otomatis terbuat.

- Copy url yang di generate pada *environment* "Tutorial-07_Mock-Server" lalu simpan pada environment "APAP" dengan nama *variable* "pilot."





- Pada collection "Tutorial-07_Web-Service", buat *folder* dengan nama "Mock" di dalam *folder* "Pilot". Lalu *duplicate request* "Get Status" dan "Get Full Status" pada collection "Tutorial-07_Mock-Server" dan pindahkan hasil duplicate tersebut ke dalam *folder* "Mock" yang baru saja dibuat (tip: *rename* agar konsisten).



- Coba jalankan kedua *request* tersebut pada *environment* "APAP", *response* yang diberikan seharusnya sesuai dengan *example* yang sudah dibuat.

6. Membuat Service Consumer

- Tambahkan dependency berikut pada pom.xml:

```
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
</dependency>
```

- Buatlah *package* baru dengan nama com.apap.tu07.rest

- Buatlah *class* bernama PilotDetail.java pada package rest dengan spesifikasi sbb.:

```
package com.apap.tu07.rest;

import java.util.Date;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;;

/**
 * PilotDetail
 */
@JsonIgnoreProperties(ignoreUnknown = true)
public class PilotDetail {
    private String status;

    @JsonProperty("valid-until")
    private Date validUntil;

    public void setStatus(String status) {
        this.status = status;
    }

    public void setValidUntil(Date validUntil) {
        this.validUntil = validUntil;
    }

    public String getStatus() {
        return status;
    }

    public Date getValidUntil() {
        return validUntil;
    }
}
```

- Buatlah *class* bernama Setting.java pada package rest dengan spesifikasi sbb.:

```
package com.apap.tu07.rest;

public class Setting {
    final public static String pilotUrl = "https://52d2de8e-5965-4472-83d8-e153b63dd3ae.mock.pstmn.io";
}
```

- Pada PilotController tambahkan kode di bawah ini (sesuaikan *package* yang perlu di-*import*):

```
@Autowired
RestTemplate restTemplate;

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

@GetMapping(value = "/status/{licenseNumber}")
public String getStatus(@PathVariable("licenseNumber") String licenseNumber) throws Exception {
    String path = Setting.pilotUrl + "/pilot?licenseNumber=" + licenseNumber;
    return restTemplate.getForEntity(path, String.class).getBody();
}

@GetMapping(value="/full/{licenseNumber}")
public PilotDetail postStatus(@PathVariable("licenseNumber") String licenseNumber) throws Exception {
    String path = Setting.pilotUrl + "/pilot";
    PilotModel pilot = pilotService.getPilotDetailByLicenseNumber(licenseNumber).get();
    PilotDetail detail = restTemplate.postForObject(path, pilot, PilotDetail.class);
    return detail;
}
```

- Coba tambahkan *request* pada Postman dan tes panggilan API yang baru Anda buat di server lokal Anda. hasilnya akan muncul sesuai dengan kembalian yang tertulis.

Latihan

1. Buatlah FlightController menjadi RestController sesuai spesifikasi sbb.:

POST add flight *{url}/flight/add*	request body: new FlightModel response : new FlightModel
PUT update flight *{url}/flight/update/{flightID}?destination={destination}&origin={origin}&time={time}* parameter dapat tidak harus lengkap	response body: "flight update success"
GET flight *{url}/flight/view/{flightNumber}*	response : FlightModel
GET all flight *{url}/flight/all*	response : List<FlightModel>
Delete flight *{url}/flight/{flightID}*	response: "flight has been deleted"

   Tambahkan pada Postman *request* dan save pada folder "Flight" di Collection "Tutorial-07_Web-Service"
2. Coba pelajari dokumentasi API yang ada pada link berikut: https://developers.amadeus.com/
   Dengan *parameter* pasti bahwa *airport* yang dicari hanya yang ada di Indonesia, Buatlah sebuah *service producer*. Jika dipanggil, akan mengembalikan daftar *airport* yang ada di suatu kota di Indonesia. Masukan dari API tersebut hanyalah nama kota. Jangan lupa masukkan *request* untuk mengakses data tersebut pada Postman.
   Hint 1: program yang Anda buat akan menggunakan *service consumer* untuk mendapatkan data dari *third party* lalu mengirimkannya lagi sebagai *service producer*.
   Hint 2: Seringkali Anda perlu membuat akun terlebih dahulu untuk mendapatkan API Key.
   Hint 3: Anda dapat menggunakan API dari provider lain selama sesuai dengan kebutuhan.

Pengumpulan

1. Penjelasan pengerjaan tutorial mengenai apa yang Anda pelajari beserta *screen capture*-nya (setiap *test/output* yang dilakukan). Pengumpulan dituliskan dalam satu file dengan format **npm_nama-lengkap-anda.pdf** dan unggah ke submission slot yang disediakan di Scele.
2. Folder tutorial-07 dikumpulkan dengan cara push ke GitHub (https://github.com/achmad-f-abka/apap).