Bab 9
REST API dengan menggunakan Spring Boot

Problem yang dihadapi ketika menggunakan framework semisal SpringMVC pada bab sebelumnya salah satunya adalah masalah menyiapkan konfigurasi aplikasi yang siap untuk dijadikan sebagai basis bagi pengembangan selanjutnya (yang fokus pada pemenahan fungsionalitas sistem alih-alih masalah teknis teknologi yang dihadapi). Menyiapkan konfigurasi seperti ini cukup menyita waktu programmer. Masalah lain yang sering adalah masalah proses deployment. Karena server yang menjalankan aplikasi terpisah dari lingkungan pengembangan (development environment), banyak langkah perlu dilalui untuk menjalankan proses deployment sistem. Perkembangan arsitektur berbasis micro-service (sebagai alternatif sistem monolitik), juga membuat pengembangan aplikasi yang lebih modular dalam scope business process (service) yang lebih kecil, memerlukan kelincahan (agility) dalam pengembangan sistem secara cepat dan tepat (rapid development).

Spring Boot memberikan alternatif solusi untuk problem-problem di atas. Pada Spring Boot konfigurasi aplikasi dilakukan secara otomatis, seiring dengan package library yang digunakan. Deployment lebih mudah, karena aplikasi Spring Boot sekaligus adalah server-nya (dengan menggunakan embeded server, Tomcat atau Jetty), yang terpaketkan sebagai aplikasi java biasa (ekstension jar). Spring Boot semakin populer seiring dengan berkembangnya implementasi arsitektur sistem berbasis micro-service.

REST Service

Representational State Transfer (REST) service, sebenarnya hanya pemanfaatan protokol HTTP yang menjadi basis bagi aplikasi Web, secara lebih konsisten. Protokol HTTP secara ringkas terdiri atas dua bagian, bagian VERB dan NOUN. Elemen VERB diwakili oleh method yang digunakan ketika mengirimkan request ke server. Sedangkan elemen NOUN diwakili oleh URL.

Method yang digunakan ketika mengirim request ke server cukup beragam yaitu GET, POST, PUT, PATCH, DELETE dan sebagaimnya. Namun umumnya aplikasi web hanya memanfaat method GET dan POST. URL sebenarnya merepresentasikan resource dalam sistem, sehingga umumnya merupakan kata benda, namun pada penerapannya URL banyak menggunakan kata kerja.

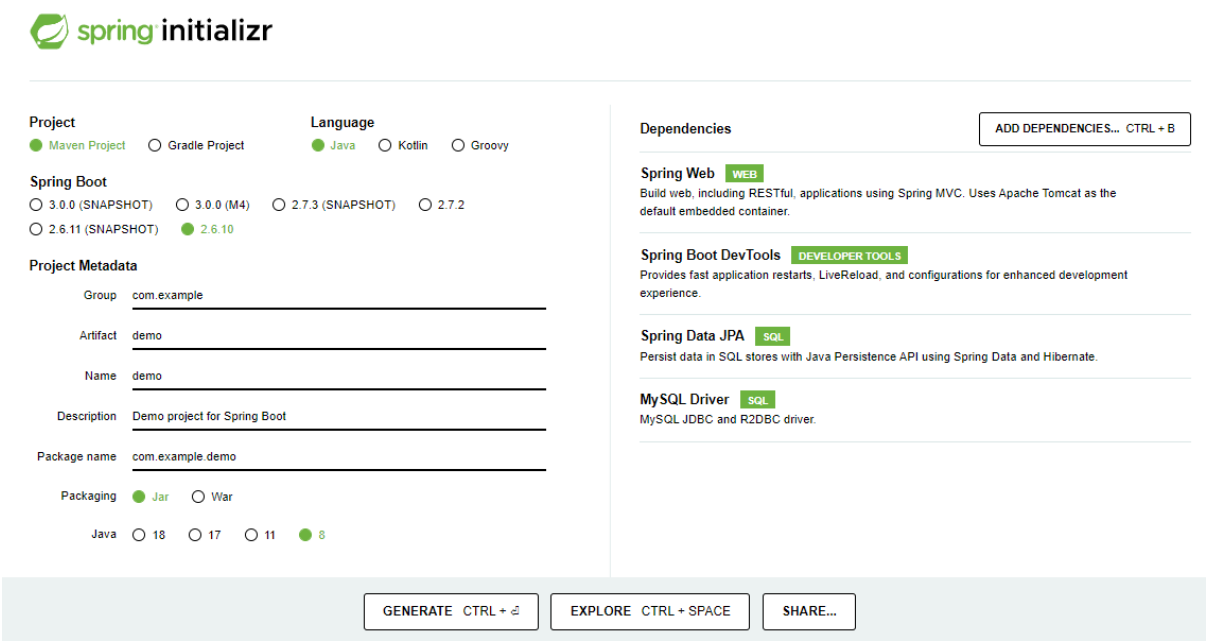
Kita dapat meringkas pemanfaatan HTTP oleh REST pada tabel berikut.

VERB	NOUN	INTERPRETASI
GET	/products	Mengambil semua data product
GET	/products/{id}	Mengambil data product dengan id tertentu
POST	/products	Menyimpan data product
PUT	/products/{id}	Melakukan update terhadap sebagian data product dengan id tertentu
PATCH	/products/{id{	Mengganti keseluruhan data product dengan id teretentu
DELETE	/products/{id}	Menghapus data product dengan id tertentu

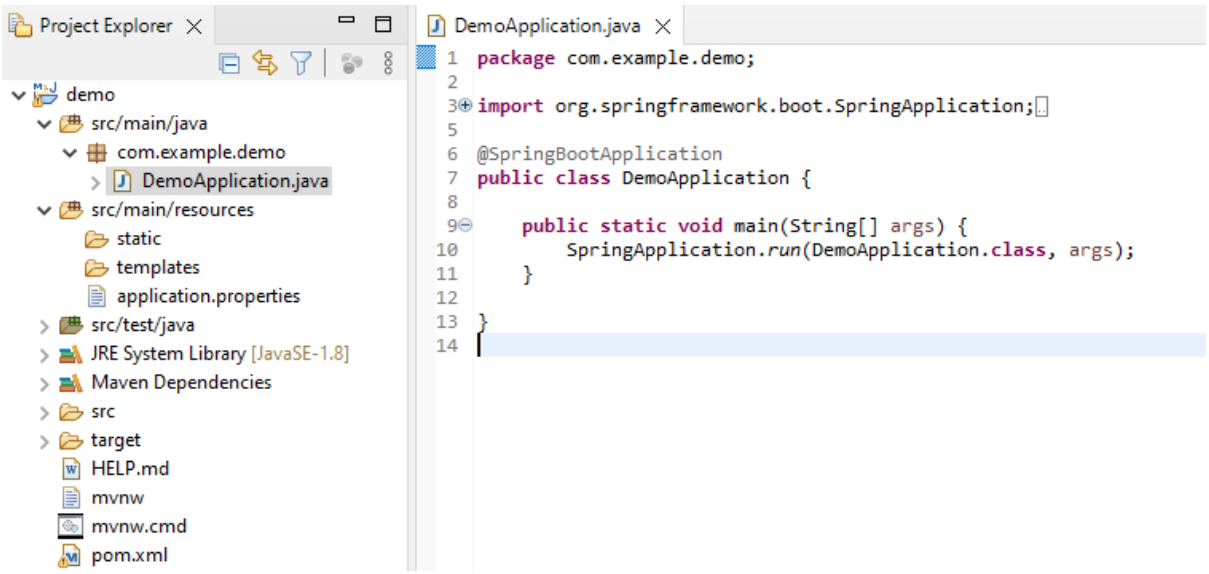
Pada REST service, data yang dikirim melalui request ke server umumnya dalam format JSON. Demikian pula data yang diterima sebagai response dari server dalam format JSON. Sedangkan status request yang dikirim diidentifikasi berdasarkan HTTP code, misal 200 OK berarti sukses, 201 Created dapat mengidikasikan proses sukses menyimpan informasi, 401 Unauthorized mengindikasikan tidak diperbolehkannya akses ke resource tertentu dan sebagainya.

Memulai Spring Boot

Kita dapat membuat kerangka aplikasi Spring Boot dengan memanfaatkan https://start.spring.io/.



Dengan menekan tombol Generate, kita akan mendapatkan sebuah file zip, dalam contoh di atas demo.zip. Silahkan extract file itu ke sebuah folder, kemudian import project yang telah dibuat oleh Spring Initailizr ini dengan menggunakan IDE (dalam hal ini kita meggunakan eclipse, *import existing maven project*).



Kita dapat menjalankan langsung aplikasi dengan me-run DemoApplication.java sebagai Java Application.

Membuat REST API

Untuk membuat REST API kita dapat membuat sebuah class yang diberikan anotasi RestController.

```
@RestController
public class HelloController {

    @GetMapping("/hello")
    public String sayHello() {
         return "Hello World";
    }
```

Anotasi @GetMapping merupakan mendefinisikan VERB untuk mengkases resource, url /hello.

Dengan membuka browser dan mengetikkan http://localhost:8080/hello pada address bar, browser akan menampilkan string Hello World yang dikirimkan oleh server.

Jika kita memiliki data yang terenkapsulasi dalam sebuah class, seperti contoh di bawah ini

```
public class EmployeeDto {
    private String firstName;
    private String lastName;
    private String type;
```

```
    public EmployeeDto(String firstName, String lastName, String type) {
        super();
        this.firstName = firstName;
        this.lastName = lastName;
        this.type = type;
    }

//Setter getter diasumsikan ada
```

Objek dari class ini dapat kita kirimkan sebagai JSON.

```
@GetMapping("/employee/info")
    public EmployeeDto employeeInfo() {
        return new EmployeeDto("Marwa", "Nurul", "Permanent");
    }
```

Dengan mengetikkan endpoint url di atas dalam browser (http://localhost:8080/employee/info) maka akan muncul informasi berikut,


```
{
    "firstName": "Marwa",
    "lastName": "Nurul",
    "type": "Permanent"
}
```


SpringBoot melakukan konversi otomatis dari object ke dalam object JSON.

Jika object berupa collection (Array, List atau Set), konversi ke JSON akan terbungkus dalam array dari object JSON.

```
@GetMapping("/employees")
    public List<EmployeeDto> employeesInfo() {
        List<EmployeeDto> empList = new ArrayList<EmployeeDto>();
        empList.add(new EmployeeDto("Marwa", "Nurul", "Permanent"));
        empList.add(new EmployeeDto("Mumtaz", "Ahmad", "Permanent"));
        empList.add(new EmployeeDto("Linna", "Sutarmila", "Permanent"));


        return empList;
    }
```

Outputnya akan menjadi sebagai berikut,

```
[
    {
        "firstName": "Marwa",
        "lastName": "Nurul",
        "type": "Permanent"
    },
    {
        "firstName": "Mumtaz",
        "lastName": "Ahmad",
        "type": "Permanent"
    },
    {
        "firstName": "Linna",
        "lastName": "Sutarmila",
        "type": "Permanent"
    }
]
```


Perhatikan tanda kurung siku pada JSON di atas. Kurung siku merepresentasikan Array pada JSON.

Integrasi Dengan Database

Untuk mengintegrasikan REST API kita dengan database relasional, cukup dengan menambahkan dependency yang diperlukan, jika kita ingin menggunakan JPA (Java Persistence API) maka dependensi yang perlu ditambahkan (dalam pom.xml) adalah Spring Data JPA (secara default menggunakan Hibernate untuk implementasinya) dan Driver Databasenya (pada bab ini kita menggunakan MySQL).

```
...
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.30</version>
</dependency>
...
```

Pada XML di atas kita menggunakan versi connector mysql yang berbeda dari default yang diberikan, karena versi java yang digunakan pada buku ini adalah Java 8.

Selanjutnya kita perlu menambahkan konfigurasi yang diperlukan dalam sebuah file, application.properties

```
server.port=8080

spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/latihan_db
```

```
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.show-sql: true
```

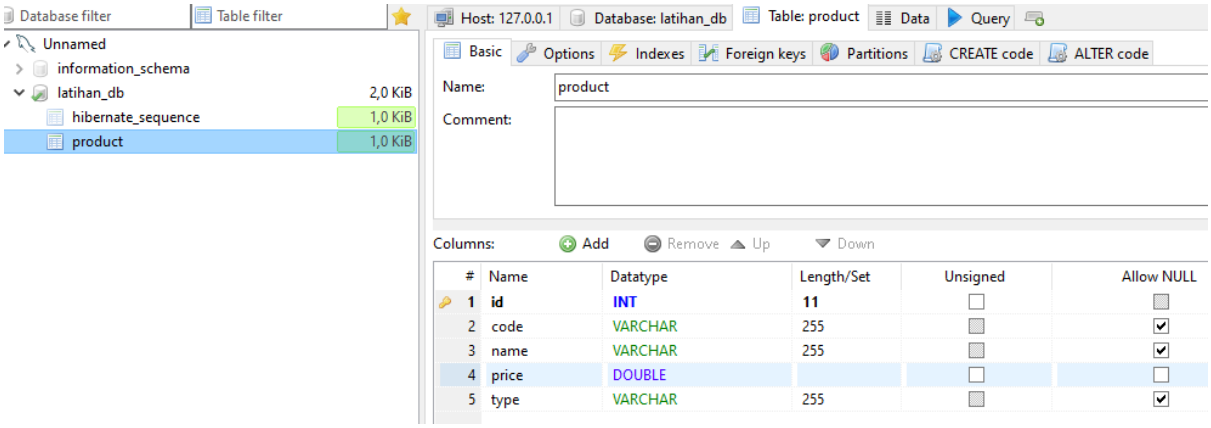
Properti ddl-auto = update, memberi informasi kepada kita jika nantinya aplikasi akan menggunakan table yang ada jika sudah ada, jika belum ada atau ada tambahan kolom yang diperlukan makan SpringBoot akan meng-create atau meng-update table itu. Properti yang lain mirip dengan informasi yang diperlukan untuk koneksi ke database pada bab sebelumnya.

Entity

Kita dapat membuat Entity sebagaimana pada bab sebelumnya. Misalnya entity product seperti di bawah ini,

```
@Entity
public class Product {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Integer id;
    private String code;
    private String name;
    private String type;
    private double price;

    public Product() {

    }

    public Product(String code, String name, String type, double price) {
         this.code = code;
         this.name = name;
         this.type = type;
         this.price = price;
    }
...
...
//Setter Getter diasumsikan ada
```


Saat aplikasi Spring Boot ini dijalankan, secara otomatis akan membuatkan table product pada database latihan_db.



Repository

Selanjutnya kita dapat membuat repository bagi entity Product. Untuk membuat repository melalui Spring Data JPA kita cukup membuat sebuah interface yang extends ke CrudRepository. Kita hanya perlu membuat interface, class yang akan mengimplementasikannya akan di-generate otomatis oleh SpringBoot saat compile.

```
public interface ProductRepository extends CrudRepository<Product, Integer>{
}
```


Dengan mendefinisikan ProductRepository sepert di atas, secara otomatasi class ini sudah mendapatkan warisan abstract method dari CrudRepository; findAll, findById, save, update, delete dan sebagainya.

Kita dapat menambahkan method lain untuk membantu kita mendefinisikan query atau perintah lain ke entity (atau database).

```
public interface ProductRepository extends CrudRepository<Product, Integer>{
    List<Product> findAll();
    Product findByCode(String code);
    List<Product> findByType(String type);

    @Query("SELECT p FROM Product p WHERE p.price > ?1")
    List<Product> searchProduct(double price);

     @Query(value="SELECT * FROM product p WHERE p.type LIKE %?1%",
       nativeQuery = true)
    List<Product> searchProductByType(String type);

}
```

Pada contoh di atas,

1. **List<Product> findAll()** mendefinisikan query untuk mengambil semua data product dalam bentuk List. Sebenarnya CrudRepository sudah menyediakan method findAll(), tetapi return-nya adalah Iterable, bukan List.

2. Product **findByCode(String code)** mendefinisikan query untuk mendapatkan sebuah product berdasarkan code-nya. Disini kita menggunakan semacam konvensi penamaan method yang akan diubah menjadi query (baik melalui JPQL atau SQL) oleh Spring Data JPA. Konvensi method ini dimulai dengan findBy… maka setelahnya adalah nama properti yang ada pada entity itu, findByCode berarti code adalah salah satu properti yang ada dalam entity Product. Kita juga dapat menggabungkan pencarian berdasarkan beberapa properti tertentu misalnya, Product **findByCodeAndType(String code, String type).**
3. **List<Product> findByType(String type)** mendefinsikan query untuk mendapatkan List product berdasarkan properti type-nya.
4. **@Query("SELECT p FROM Product p WHERE p.price > ?1")** yang ditempatkan di atas method **List<Product> searchProduct(double price)**, menggunakan JPQL (Java Persistence Query Language) untuk query mendapatkan List product dengan harga lebih besar dari nilai tertentu. Product p pada JPQL di atas adalah nama Entity-nya bukan nama table.
5. **@Query(value="SELECT * FROM product p WHERE p.type LIKE %?1%", nativeQuery = true)** yang ditempatkan pada method **List<Product> searchProductByType(String type)**, menggunakan SQL (Structured Query Language) untuk mendapatkan List product dengan type tertentu. Perhatikan, product p pada SQL di atas adalah nama table-nya bukan Entity.

Service

Class service biasanya digunakan untuk membuat proses bisnis (dalam sebuah method) yang melibatkan akses ke beberapa repository, juga digunakan untuk mengontrol proses transaksi ke repository. Pada contoh ini kita hanya menggunakan service untuk mengontrol transaksi.

```
@Service
public class ProductService {

    @Autowired
    ProductRepository repo;

    @Transactional
    public void save(Product product) {
         repo.save(product);
    }
}
```

Rest Controller

Kita mendefinisikan Rest Controller untuk mengimplementasikan table REST berikut,

VERB	NOUN	INTERPRETASI
POST	/products	Menyimpan data product
GET	/products	Mengambil semua data product
GET	/products/{code}	Mengambil data product dengan code tertentu

```
@RestController
@RequestMapping(path = "/v1")
public class ProductController {

    @Autowired
    ProductRepository productRepo;

    @Autowired
    ProductService productService;

    @PostMapping("/products")
    public ResponseEntity<Product> saveProduct(@RequestBody ProductDto productDto) {
         Product product = new Product(productDto.getCode(), productDto.getName(),
                                       productDto.getType(), productDto.getPrice());
         productService.save(product);
         return new ResponseEntity<Product>(product, HttpStatus.OK);
    }
...
...
```


Pada @RequestMapping di bawah @RestController kita tambahkan /v1 untuk mengontrol versi dari API kita. ProductController dapat langsung mengakses ProductRepository maupun ProductService (kita tidak membuat aturan layer yang ketat).

Disini dibedakan fungsi dari Product sebagai Entity dan ProductDto sebagai model untuk view (yang akan ditampilkan atau digunakan untuk menerima input dari user). ProductDto adalah view model, sedangkan Product adalah business model.

(1) Menyimpan data product

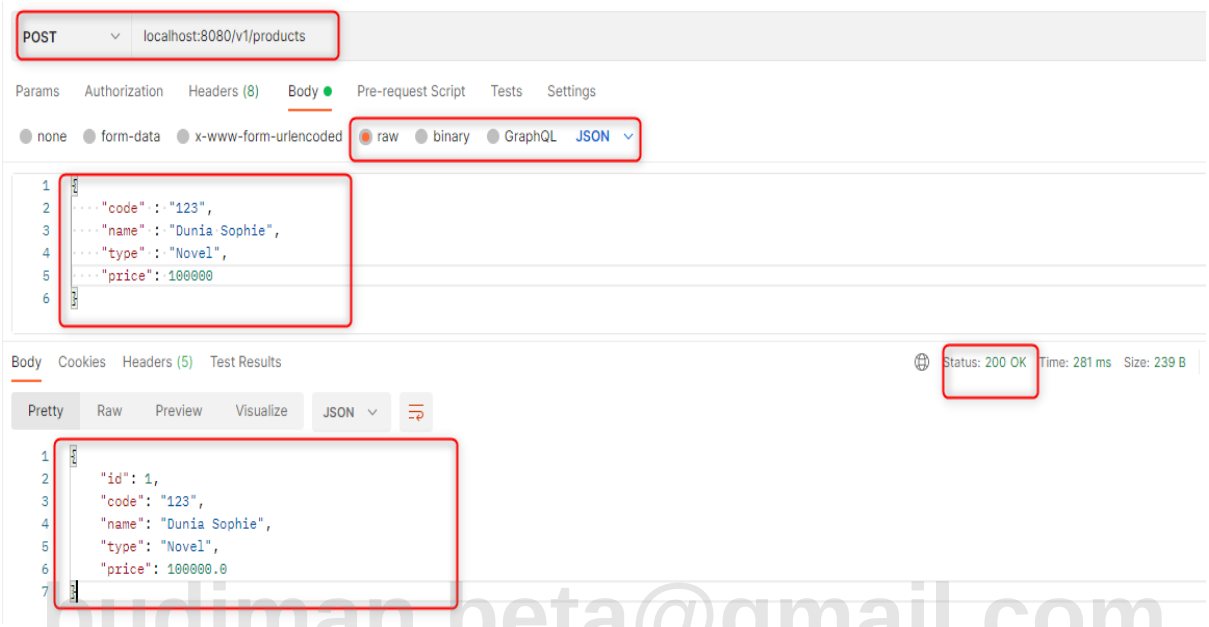
```
    @PostMapping("/products")
    public ResponseEntity<Product> saveProduct(@RequestBody ProductDto productDto) {
         Product product = new Product(productDto.getCode(), productDto.getName(),
                                       productDto.getType(), productDto.getPrice());
         productService.save(product);
         return new ResponseEntity<Product>(product, HttpStatus.OK);
    }
```

@PostMapping(“/products”)	VERB : POST, NOUN : /products
saveProduct(@RequestBody Product product)	Menerima raw data (JSON) dari request

ResponseEntity<Product>	Mengembalikan JSON object dari product yang disimpan, dengan HTTP status tertentu.

Method saveProduct di atas akan mengembalikan HTTP status OK (200), dan object JSON dari objek yang disimpan.

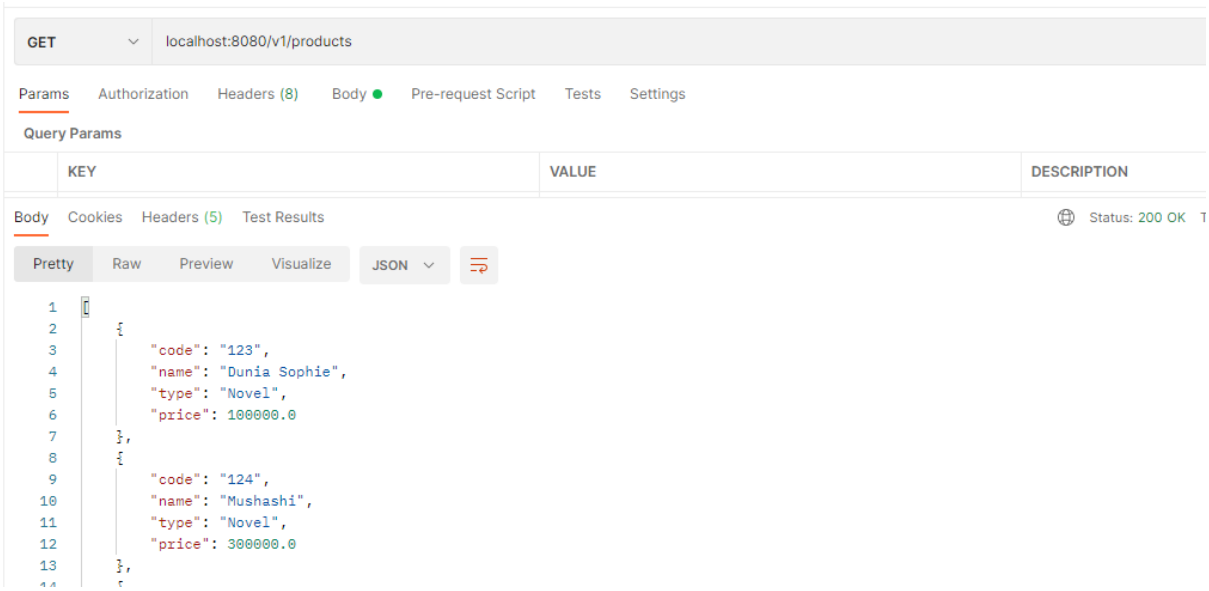
Gunakan PostMan untuk melakukan testing.



(2) Mengambil semua data product

Kita dapat menambahkan method berikut pada ProductController untuk mengambil semua data product.

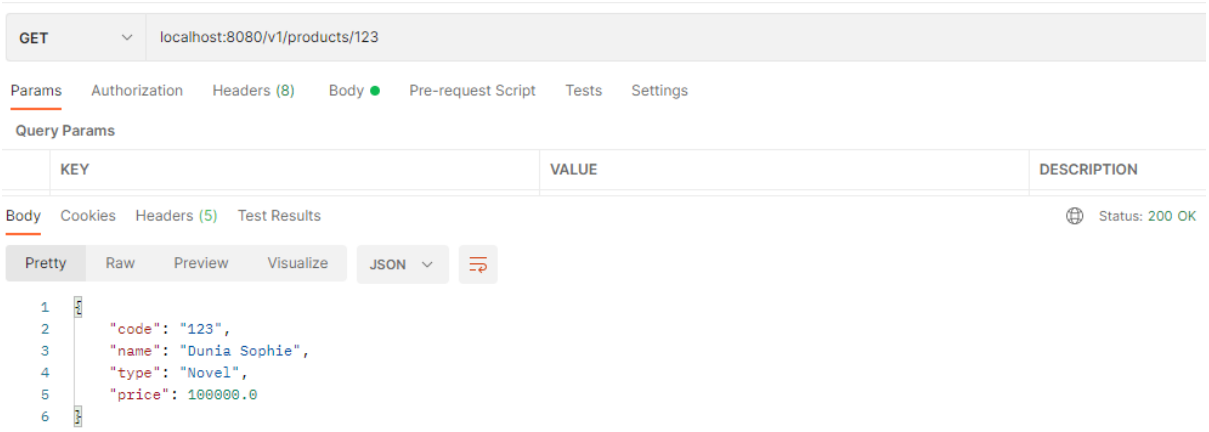
```
@GetMapping("/products")
public ResponseEntity<List<ProductDto>> getAllProduct(){
     List<Product> products =  productRepo.findAll();
     List<ProductDto> productDtoList = new ArrayList<ProductDto>();
     for(Product p : products) {
          productDtoList.add(new ProductDto(p.getCode(), p.getName(),
                         p.getType(), p.getPrice()));
     }
     return new ResponseEntity<List<ProductDto>>(productDtoList,
                         HttpStatus.OK);
}
```



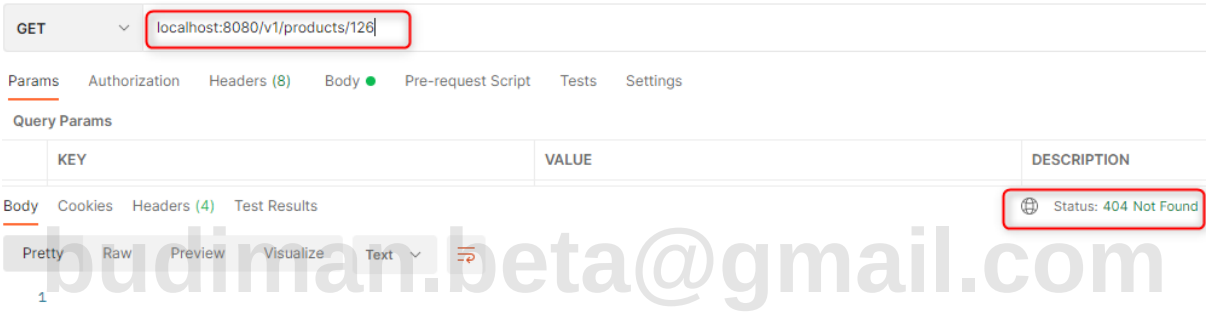
(3) Mengambil data product dengan code tertentu

Berikut code yang dapat digunakan untuk mengambil data product berdasarkan code tertentu,

```
@GetMapping("/products/{code}")
    public ResponseEntity<ProductDto> getProductByCode(
                                        @PathVariabel("code") String code) {
        ProductDto producDto = null;

            try {
                Product product = productRepo.findByCode(code);
                producDto = new ProductDto(product.getCode(),product.getName(),
                          product.getType(), product.getPrice());

            }catch(Exception e) {

            }
            if(producDto != null) {
                return new ResponseEntity<ProductDto>(producDto, HttpStatus.OK);
            }else {
                return new ResponseEntity<ProductDto>(HttpStatus.NOT_FOUND);
            }

    }
```



Jika data ditemukan, HTTP Status adalah OK (200), dengan menampilkan JSON object dari data product yang ditemukan. Sedangkan jia data tidak ditemukan HTTP Status adalah NOT FOUND (404).



Source code ProductController secara lengkap adalah sebagai berikut,

```
@RestController
@RequestMapping(path = "/v1")
public class ProductController {

    @Autowired
    ProductRepository productRepo;

    @Autowired
    ProductService productService;

    @PostMapping("/products")
    public ResponseEntity<Product> saveProduct(@RequestBody ProductDto productDto) {
         Product product = new Product(productDto.getCode(), productDto.getName(),
                                       productDto.getType(), productDto.getPrice());
         productService.save(product);
         return new ResponseEntity<Product>(product, HttpStatus.OK);
    }

    @GetMapping("/products")
    public ResponseEntity<List<ProductDto>> getAllProduct(){
         List<Product> products =  productRepo.findAll();
         List<ProductDto> productDtoList = new ArrayList<ProductDto>();
         for(Product p : products) {
              productDtoList.add(new ProductDto(p.getCode(), p.getName(),
                        p.getType(), p.getPrice()));
```

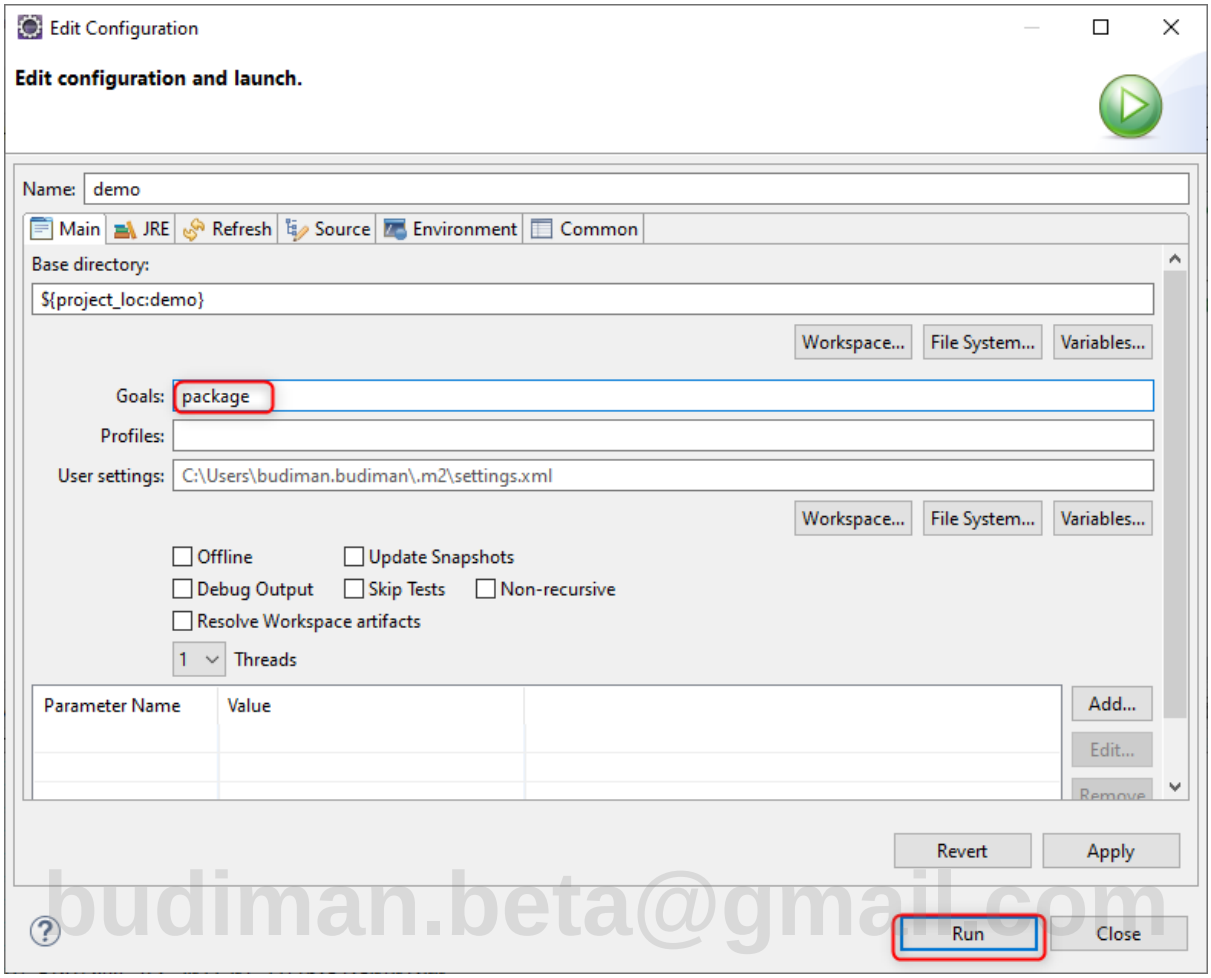
```
        }
        return new ResponseEntity<List<ProductDto>>(productDtoList, HttpStatus.OK);
    }

    @GetMapping("/products/{code}")
    public ResponseEntity<ProductDto> getProductByCode(
                          @PathVariabel("code") String code) {
        ProductDto producDto = null;

        try {
            Product product = productRepo.findByCode(code);
            producDto = new ProductDto(product.getCode(),product.getName(),
                    product.getType(), product.getPrice());

        }catch(Exception e) {

        }
        if(producDto != null) {
            return new ResponseEntity<ProductDto>(producDto, HttpStatus.OK);
        }else {
            return new ResponseEntity<ProductDto>(HttpStatus.NOT_FOUND);
        }

    }

}
```

Deployment

Deployment aplikasi Spring Boot cukup mudah, karena Spring Boot dipaketkan dalam betuk file jar (lebih tepatnya fat jar, karena ukurannya cukup besar). Untuk mendapatkan file jar ini kita bisa meminta maven untuk melakukan build dan membuat package bagi aplikasi ini. Perintah maven-nya adalah :

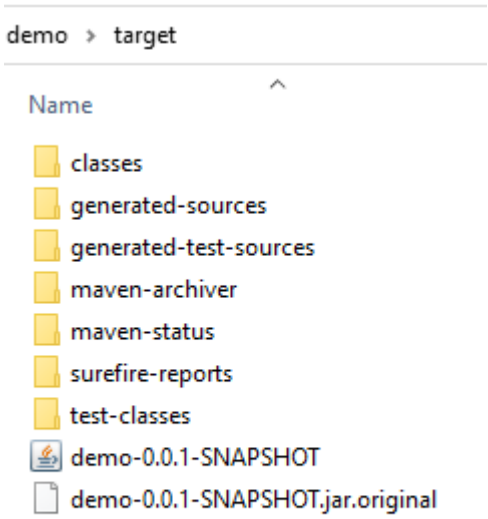
maven build package

Dengan menggunakan Eclipse kita dapat menjalan Run As > 3 Maven Build. Jika ini pertama kali kita menjalankan Maven Build maka akan muncul,



Isi package pada goal, kemudian klik Run.

File jar hasil kompilasi ada pada folder **target** pada folder project.



Anda dapat meng-copy file jar demo-0.0.1-SNAPSHOT ke satu folder lain, kemudian menjalankannya dengan perintah :

java -jar demo demo-0.0.1-SNAPSHOT.jar

Konfigurasi aplikasi dapat diubah dengan menempatkan file application.properties dalam folder yang sama, misal dengan mengubah port-nya menjadi 8181.

```
server.port=8181

spring.jpa.hibernate.ddl-auto=update
spring.datasource.url=jdbc:mysql://localhost:3306/latihan_db
spring.datasource.username=root
spring.datasource.password=
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.show-sql: true
```

Folder Demo