



# Back End Development 1 ○

---

+

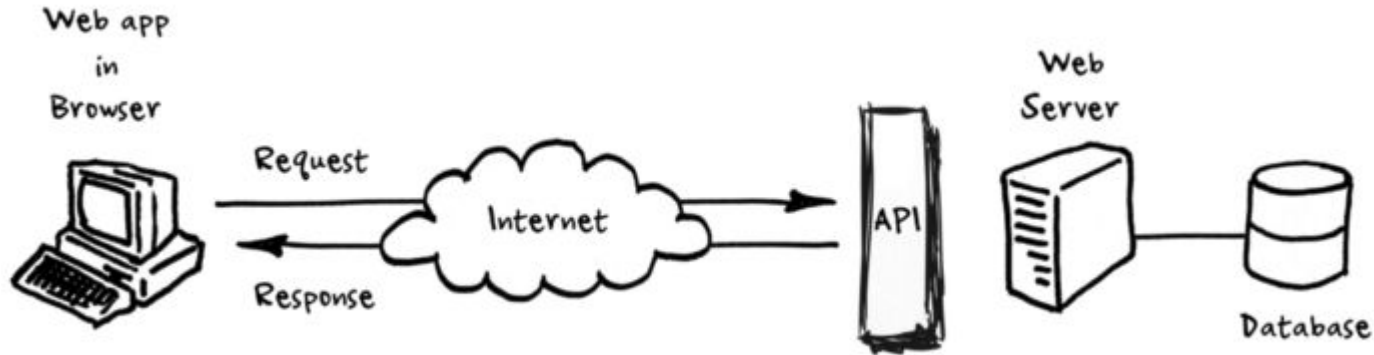
## Sesi 17



# **Springboot -<sup>+</sup> Restful Web Services/REST Template**

## Springboot - Restful Web Services/REST Template - Sesi 17

# Concept of API, Why & Scope Implementation



Source : <https://medium.com/@perrysetgo/what-exactly-is-an-api-69f36968a41f>

**API = Application Programming Interface**

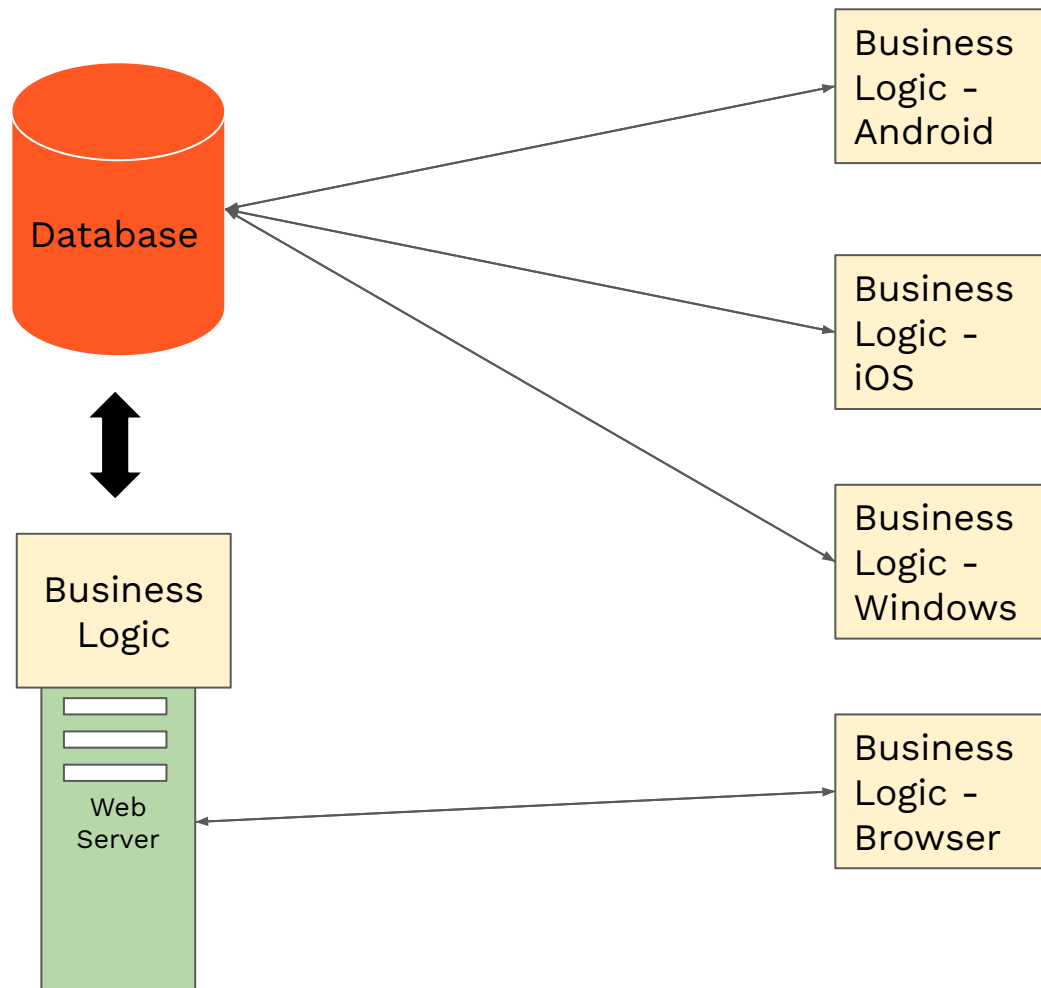
“interface yang digunakan oleh satu layanan dengan yang lainnya untuk saling 'berkomunikasi' “

**An API is not a database. It is an access point to an app that can access a database.**



**HACKTIV8**

# Analogi API



API sama halnya dengan menu makanan yang tersedia di sebuah restaurant, tertera dengan lengkap di dalam menu tersebut list makanan beserta bahan-bahannya. Ketika kita memesan menu pilihan, koki akan menyiapkan pesanan tersebut, kita cukup menunggu hasilnya. Ketika pesanan selesai, pelayan akan mengantarkan pesanan tersebut ke hadapan kita.

Begitu pula dengan API, ketika kita ingin menampilkan data dari sebuah database, kita hanya perlu menggunakan daftar fungsi yang telah ada dan menunggu hasilnya. Semua proses tersebut akan dikerjakan oleh API. Kita tidak perlu bersusah payah mengetahui bagaimana proses tersebut terjadi, *cukup terima jadi*.

Dengan kata lain, jika kita mempunyai sebuah aplikasi (core), kita dapat mendistribusikannya kembali ke platform/aplikasi lain menggunakan bantuan API.

Tentunya, penggunaan API tidak terbatas sampai di situ saja. Dengan semakin berkembangnya teknologi, para developer juga menggunakan API untuk komunikasi dua arah di aplikasi yang sama, antara front-end dan back-end yang menggunakan bahasa pemrograman berbeda seperti: vuejs sebagai front-end dan laravel sebagai backend, atau reactjs sebagai front-end dan django sebagai back-end.



### Why API ?

Jika kita seorang pengembang perangkat lunak, tentunya akan sangat merasakan perbedaan yang signifikan. Terlebih apabila aplikasi tersebut memiliki kebutuhan yang complex dan berskala besar serta cross platform.

Contoh sederhana penggunaan API adalah, ketika kita memiliki sebuah aplikasi berbasis web, agar aplikasi tersebut dapat berjalan di platform lain seperti iOS, Android ataupun Desktop, kita dapat menggunakan sebuah API. Sehingga aplikasi tersebut dapat berjalan di manapun tanpa membuat versi lain untuk masing-masing platform.

Penggunaan API memberikan benefit yang besar untuk seorang developer, karena dengan bantuan API, developer tidak perlu membuat banyak aplikasi inti (core) untuk masing-masing platform yang berbeda. Aplikasi inti tersebut dapat kembali digunakan dengan bantuan API sehingga lebih menghemat waktu pengembangan project tersebut.

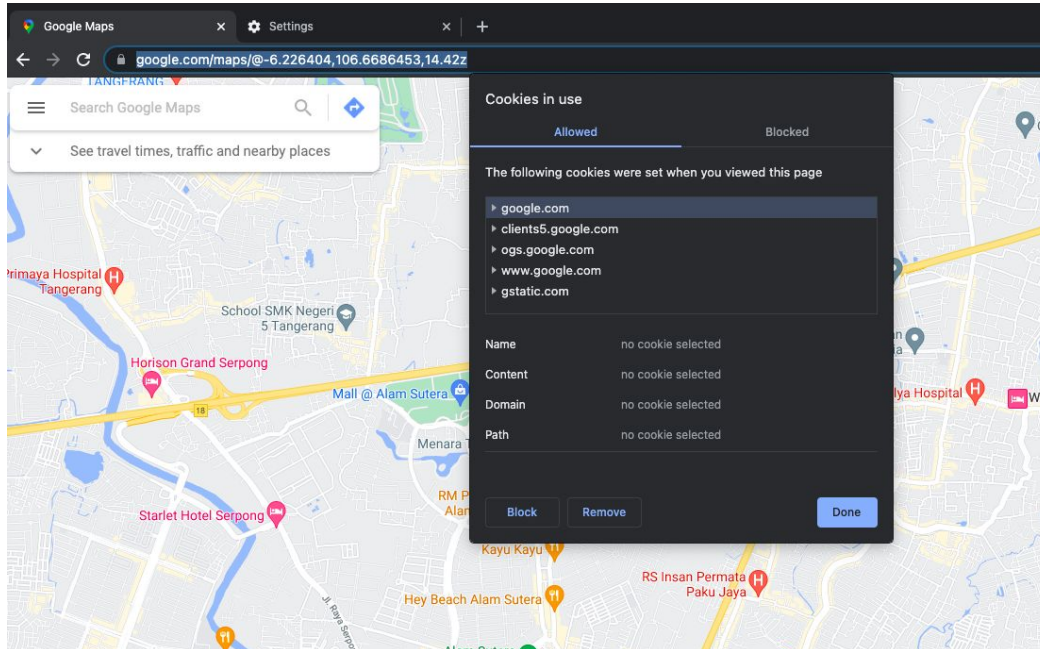


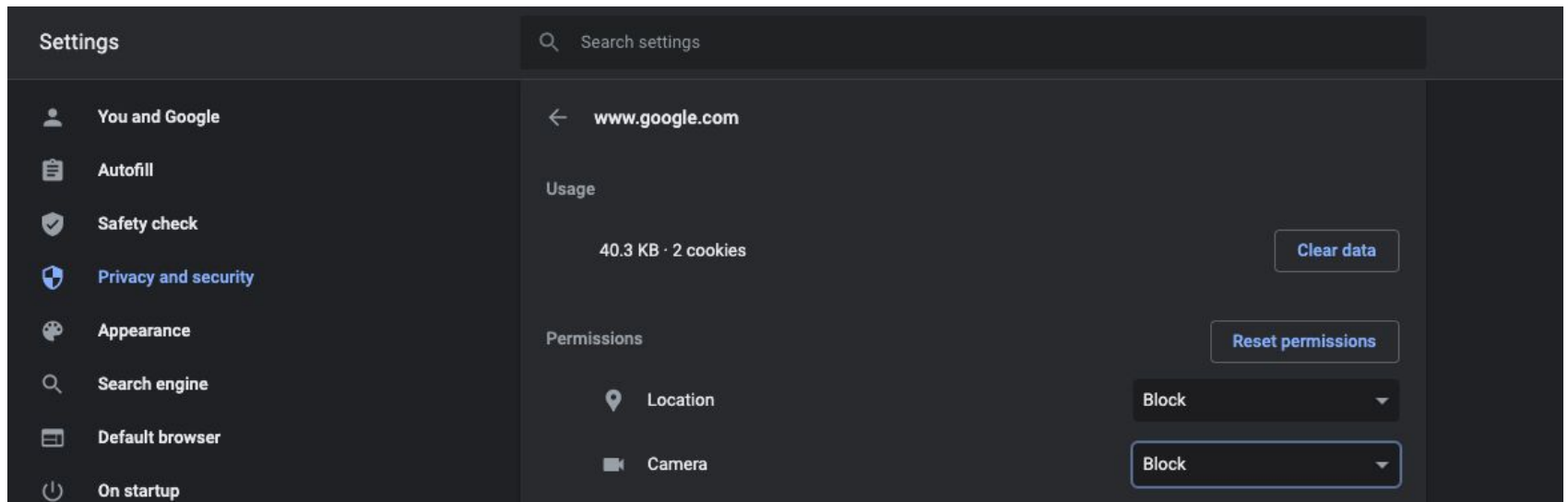
# Springboot - Restful Web Services/REST Template - Sesi 17

## Contoh Penggunaan API

Lalu seperti apa contoh penggunaan API yang bisa kita temui?

Cukup mudah, yaitu ketika kita mengunjungi sebuah web / situs salah satunya adalah Google Maps, terkadang situs tersebut akan meminta izin untuk mengakses gps pada perangkat kita melalui browser. Hal tersebut adalah contoh mudah penggunaan API yang sering kita temui.



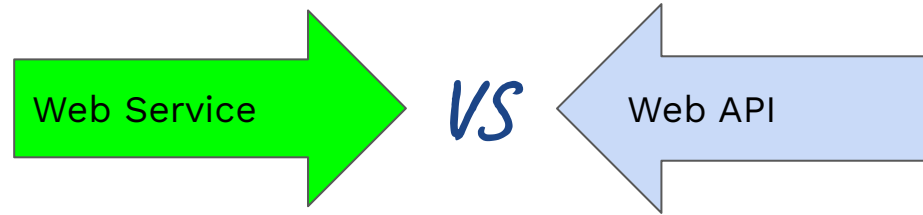


Gambar di atas adalah salah satu contoh penggunaan API pada Google Maps yang telah kita blokir.



## Springboot - Restful Web Services/REST Template - Sesi 17

# Perbedaan Web Services dan Web API



Lalu apa sih perbedaan antara *web API* dan *web service*?

1. *Web service* memfasilitasi untuk melakukan interaksi antara dua perangkat atau aplikasi melalui jaringan. Sedangkan *API* bertindak sebagai penghubung antara dua aplikasi berbeda sehingga bisa berkomunikasi satu sama lain baik dengan ataupun tanpa jaringan.
2. Semua *web service* menggunakan *API* tapi tidak semua *API* digunakan sebagai *web service*
3. *Web service* selalu membutuhkan jaringan untuk pengoperasiannya sedangkan *API* tidak selalu memerlukan jaringan untuk operasinya.
4. *Web service* hanya menggunakan 3 *style* yaitu *SOAP*, *REST*, atau *XML-RPC* untuk berkomunikasi sedangkan *API* dapat menggunakan *style* apapun.

## REST API

REST merupakan singkatan dari *Representational State Transfer*. Secara singkat REST adalah cara kita untuk menggunakan *resource* (fungsi/method) yang ada di sebuah *server* dengan mengakses *url* yang telah disediakan.

Cara mengaksesnya tentu dengan menggunakan HTTP (*Hyper Text Transfer Protocol*) dengan *method* (*http verb*) yang umum digunakan yaitu:

- GET, untuk membaca *resource* (data).
- POST, untuk membuat *resource* baru (data baru).
- DELETE, tentu untuk menghapus *resource* (data).
- PUT, untuk merubah *resource* (data).

Dan yang perlu diingat, bahwa REST ini adalah *stateless*, artinya tidak ada *state* di dalamnya. Misalnya tidak ada penggunaan *session*. Karena sifatnya, klien hanya meminta ke *server* dan *server* akan memberikan *response*-nya. **titik**. Hanya sampai situ saja. Sehingga untuk proses otentikasinya, kita tidak dapat menggunakan *session*.

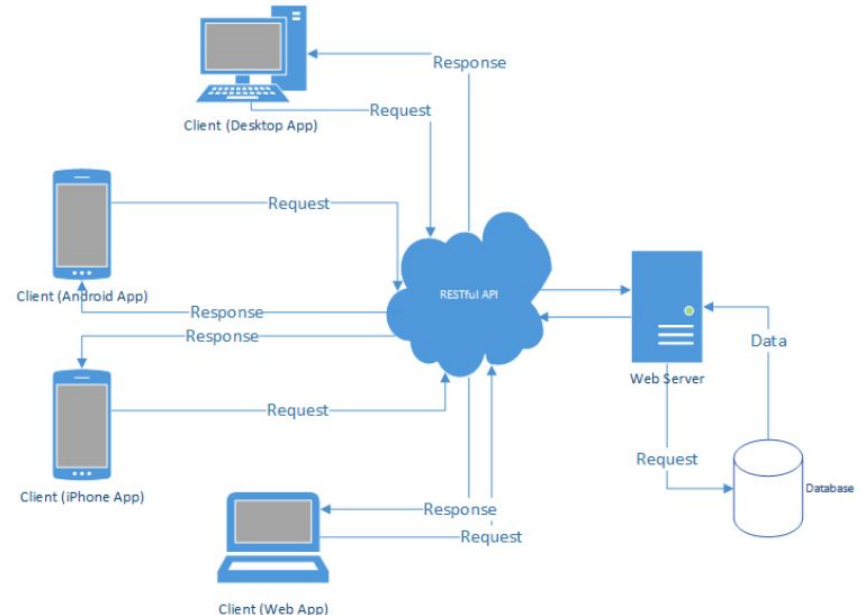
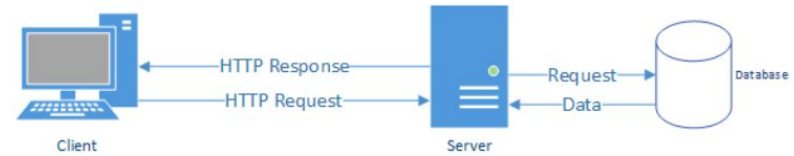


## Why we need REST API ?

Pertama harus ada sebuah REST server yang akan menyediakan resource/data. Sebuah REST client akan membuat HTTP request ke server melalui sebuah global ID atau URIs dan server akan merespon dengan mengirimkan balik sebuah HTTP response sesuai yang diminta client.

### Komponen Rest API

- HTTP method seperti GET, POST, PUT, DELETE dll sesuai dengan tugasnya masing-masing
- URI (*Uniform Resource Identifier*) untuk mengetahui lokasi data di server
- HTTP Version, seperti HTTP v1.1
- Request Header, berisi metadata seperti Authorization, tipe client dan lain
- Request Body, data yang diberikan client ke server seperti URI params



# Komponen Restful API

## REST API REQUEST COMPONENT

- HTTP method seperti GET, POST, PUT, DELETE dll sesuai dengan tugasnya masing-masing
- URI (*Uniform Resource Identifier*) untuk mengetahui lokasi data di server
- HTTP Version, seperti HTTP v1.1
- Request Header, berisi metadata seperti Authorization, tipe client dan lain
- Request Body, data yang diberikan client ke server seperti URI params

## REST API RESPONSE COMPONENT

- Response Code, status server terhadap request yang diminta seperti 200, 401, 404 dan lainnya.
- HTTP Version
- Response Header yang berisi meta-data seperti content-type, cache tag dan yang lainnya.
- Response Body, data/resource yang diberikan oleh server baik itu berupa text, json ataupun xml



# Prinsip Restful API

Ada 6 batasan panduan yang mendefinisikan arsitektur REST, yaitu:

- **Uniform Interface:** Antarmuka komponen harus sama. Ini berarti menggunakan sekitar URI (*Uniform Resource Identifier*) untuk mengidentifikasi sumber daya—dengan kata lain, path yang dapat dimasukkan ke bilah lokasi browser.
- **Client-Server:** Ada pemisahan kekhawatiran antara server, yang menyimpan dan memanipulasi data, dan klien, yang meminta dan menampilkan respon.
- **Stateless Interactions:** Semua informasi tentang setiap permintaan terkandung dalam setiap permintaan individu dan tidak tergantung pada status session.
- **Cacheable:** Klien dan server dapat menyimpan sumber daya.
- **Layered System:** Klien dapat dihubungkan ke server akhir, atau lapisan menengah seperti load-balancer.
- **Code on Demand (Opsional):** Seorang klien dapat mengunduh kode, yang mengurangi visibilitas dari luar.

## Request dan Response



# Request Method

Ada empat metode HTTP utama, juga disebut sebagai kata kerja HTTP, yang biasanya digunakan untuk berinteraksi dengan API web. Metode-metode ini menentukan tindakan yang akan dilakukan dengan sumber daya apa pun yang diberikan.

Metode request HTTP secara longgar sesuai dengan paradigma CRUD, yang merupakan singkatan dari Create, Update, Read, Delete. Meskipun CRUD mengacu pada fungsi yang digunakan dalam operasi basis data, kita dapat menerapkan prinsip-prinsip desain tersebut ke kata kerja HTTP di RESTful API.

## Kode Response

Setelah permintaan melewati dari klien ke server, server akan mengirim kembali respon HTTP, yang akan mencakup metadata tentang respon yang dikenal sebagai header, serta badan. Bagian pertama dan terpenting dari respon adalah kode status, yang menunjukkan apakah permintaan berhasil, jika ada kesalahan, atau jika tindakan lain harus diambil.

Request	Respon
GET	200 (OK)
POST	201(Created)
PUT	200 (OK)
DELETE	200 (OK), 202 (Accepted), or 204 (No Content)



Kita akan mencoba mengetest aplikasi menggunakan REST API Client yang gratis yaitu Postman, silahkan bisa download nya disini <https://www.getpostman.com/downloads/> dan sesuaikan dengan sistem operasi laptop kalian.



**HACKTIV8**



# REST API With Springboot

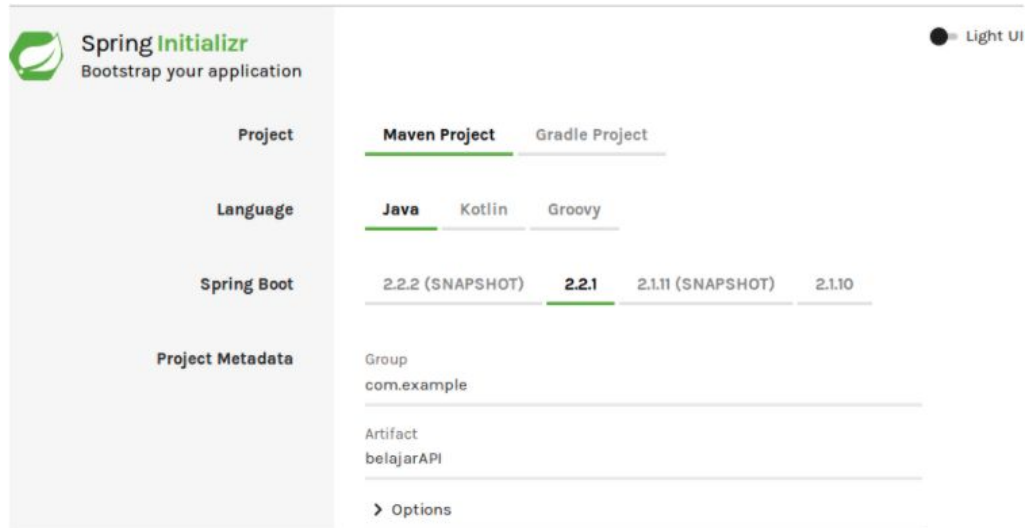


## Springboot - Restful Web Services/REST Template - Sesi 17

# Membuat REST API

Untuk memulai project, kita bisa kunjungi <https://start.spring.io/>

- Pilih Maven Project untuk build toolnya
- Pilih Java untuk bahasa pemrogramannya
- Ubah nama artifact sesuai keinginanmu, misalkan dalam project ini saya namakan belajarAPI
- Untuk project sederhana ini tambahkan dependencies Spring Web
- Generate Project, lalu download artifact yang sudah tergenerate.



The screenshot shows the Spring Initializr web application bootstrap form. The form is titled "Spring Initializr" with the subtitle "Bootstrap your application". It has a "Light UI" toggle in the top right corner. The form is divided into four sections: "Project", "Language", "Spring Boot", and "Project Metadata".

- Project:** "Maven Project" is selected over "Gradle Project".
- Language:** "Java" is selected over "Kotlin" and "Groovy".
- Spring Boot:** "2.2.1" is selected over "2.2.2 (SNAPSHOT)", "2.1.11 (SNAPSHOT)", and "2.1.10".
- Project Metadata:**
  - Group:** "com.example"
  - Artifact:** "belajarAPI"
  - Options:** A dropdown arrow is visible.



Dependencies

Q

1 selected

Search dependencies to add

Web, Security, JPA, Actuator, Devtools...

Selected dependencies

**Spring Web**

Build web, including RESTful, applications using Spring MVC.  
Uses Apache Tomcat as the default embedded container.

Generate - Ctrl + ⌘

Explore - Ctrl + Space

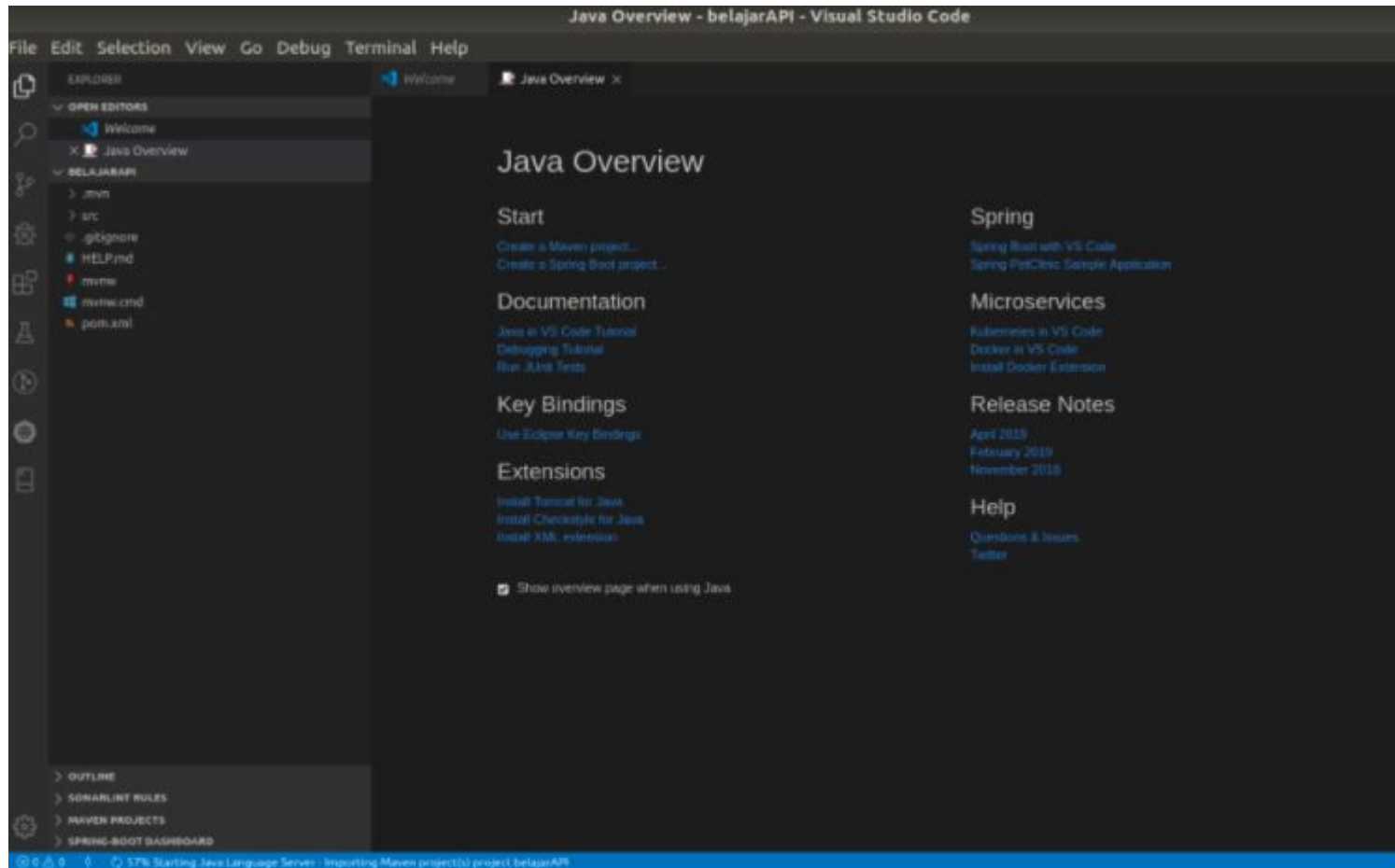
Share...

© 2013-2019 Pivotal Software

start.spring.io is powered by

[JHipster](#) and [Pivotal Web Services](#)

Setelah berhasil didownload, extract dokumen tersebut kemudian buka dengan IDE kita. Kali ini kita akan menggunakan Visual Studio Code untuk membuka project belajarAPI.



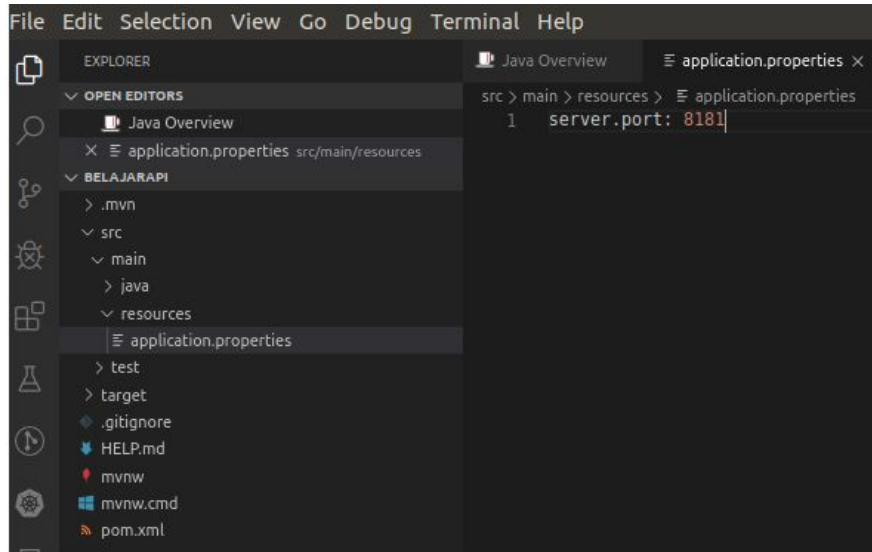
HACKTIV8

## Springboot - Restful Web Services/REST Template - Sesi 17

# Setting Environment di application.properties

Kita dapat mengatur konfigurasi project kita melalui application.properties yang dapat kita temukan dalam folder src → main → resources.

Untuk project kali ini kita cukup mengatur server port kita saja ke port 8181, anda bisa menggunakan port yang lain secara bebas selama tidak dipakai oleh aplikasi lain yang sedang running.



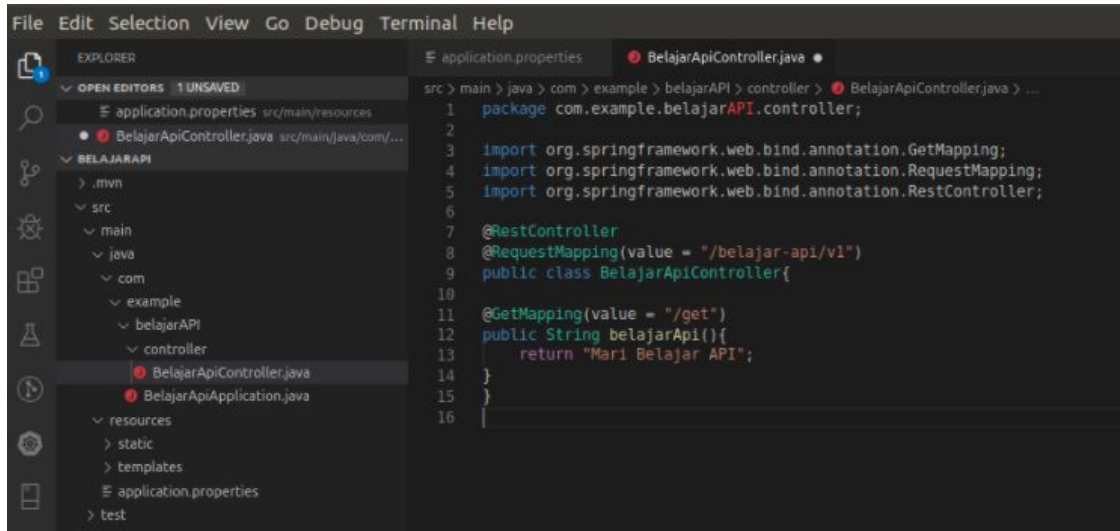
## Springboot - Restful Web Services/REST Template - Sesi 17

### Controllers

Langkah selanjutnya kita akan membuat controller sederhana untuk menempatkan end point API yang akan kita buat.

Adapun API yang akan kita buat adalah API sederhana yang akan mereturn “Mari Belajar API” ketika dihit dengan metode GET.

- Buat package/folder controller di dalam package belajarAPI
- Buat class baru dengan nama BelajarApiController.java



```
File Edit Selection View Go Debug Terminal Help
EXPLORER
OPEN EDITORS 1 UNSAVED
application.properties src/main/resources
BelajarApiController.java src/main/java/com/...
BELAJARAPI
.mvn
src
main
java
com
example
belajarAPI
controller
BelajarApiController.java
BelajarApiApplication.java
resources
static
templates
application.properties
test

src > main > java > com > example > belajarAPI > controller > BelajarApiController.java > ...
1 package com.example.belajarAPI.controller;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController
8 @RequestMapping(value = "/belajar-api/v1")
9 public class BelajarApiController{
10
11 @GetMapping(value = "/get")
12 public String belajarApi(){
13     return "Mari Belajar API";
14 }
15 }
16
```

- Anotasi `@RestController` untuk menjadikan class java kita sebagai Rest Controller.
- Anotasi `@RequestMapping` lalu masukkan value berupa end point yang akan kita jadikan API kita. Pada sesi ini kita membuat API dengan url endpoint `localhost:8181/belajar-apiv1/get`.
- Anotasi `@GetMapping` untuk membuat API dengan GET method. Untuk method POST dapat menggunakan `@PostMapping`.  
Tambahkan value untuk mengkhususkan API yang kita buat agar memiliki end point yang kita harapkan.
- Dalam API sederhana ini kita berikan return value String dengan pesan “Mari Belajar API”.

Untuk menjalankan program jalankan command `mvn spring-boot:run` pada direktori project anda melalui terminal.

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
988 seconds (JVM running for 1.195)
2019-11-27 11:44:00.813 INFO 13589 --- [nio-8181-exec-2] o.a.c.c.C.[Tomcat].
et 'dispatcherServlet'
2019-11-27 11:44:00.813 INFO 13589 --- [nio-8181-exec-2] o.s.web.servlet.Dis
vlet'
2019-11-27 11:44:00.817 INFO 13589 --- [nio-8181-exec-2] o.s.web.servlet.Dis
^C2019-11-27 11:44:14.632 INFO 13589 --- [extShutdownHook] o.s.s.concurrent.
plicationTaskExecutor'
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 45.962 s
[INFO] Finished at: 2019-11-27T11:44:14+07:00
[INFO] -----
~/Documents/Exercise/belajarAPI$ mvn spring-boot:run
```

Apabila berhasil running akan muncul tampilan seperti ini :

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
81 (http)
2019-11-27 13:32:07.444 INFO 15063 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2019-11-27 13:32:07.444 INFO 15063 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.27]
2019-11-27 13:32:07.488 INFO 15063 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2019-11-27 13:32:07.488 INFO 15063 --- [main] o.s.web.context.ContextLoader : Root WebApplicationContext: initialization completed in 518 ms
2019-11-27 13:32:07.590 INFO 15063 --- [main] o.s.s.concurrent.ThreadPoolTaskExecutor : Initializing ExecutorService 'applicationTaskExecutor'
2019-11-27 13:32:07.680 INFO 15063 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8181 (http) with context path ''
2019-11-27 13:32:07.683 INFO 15063 --- [main] c.e.belajarAPI.BelajarApiApplication : Started BelajarApiApplication in 0.993 seconds (JVM running for 1.201)
```

Untuk mencoba API yang sudah kita buat anda bisa menggunakan Postman, atau untuk case kita kali ini karena hanya berupa GET API kita bisa langsung menggunakan browser kita dengan mengetikkan alamat end point API yang sudah kita buat yakni localhost:8181/belajar-api/v1/get dan akan mengembalikan pesan “Mari Belajar API”.



Mari Belajar API



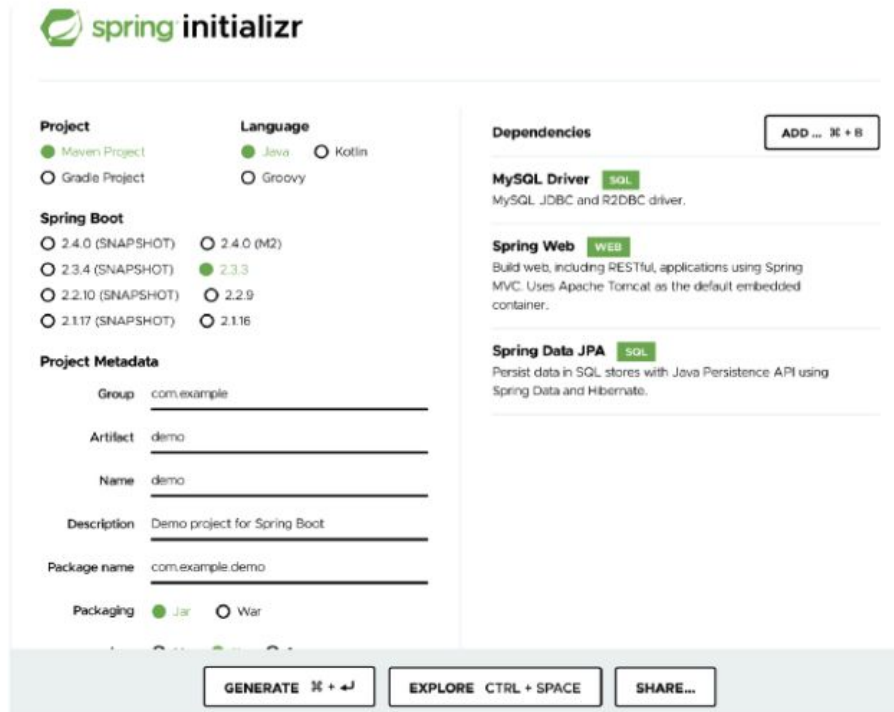
# **REST API With Springboot with MySQL**



## Springboot - Restful Web Services/REST Template - Sesi 17

# REST API with MySQL

Di sesi ini kita akan membuat REST API using HTTP Methods untuk operasi CRUD ( Create, Retrieve, Update, & Delete) dengan springboot menggunakan database MySQL.



The image shows the Spring Initializr web form for generating a Spring Boot project. The form is divided into several sections:

- Project:** Radio buttons for ☒ Maven Project and ☐ Gradle Project.
- Language:** Radio buttons for ☒ Java and ☐ Kotlin, and ☐ Groovy.
- Spring Boot:** Radio buttons for versions 2.4.0 (SNAPSHOT), 2.3.4 (SNAPSHOT), 2.2.10 (SNAPSHOT), 2.1.17 (SNAPSHOT), 2.4.0 (M2), ☒ 2.3.3, 2.2.9, and 2.1.16.
- Project Metadata:** Text input fields for Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), and Package name (com.example.demo). A Packaging section has ☒ Jar and ☐ War.
- Dependencies:** A list of dependencies with checkboxes: ☒ MySQL Driver (SQL), ☒ Spring Web (WEB), and ☒ Spring Data JPA (SQL). An "ADD ... 36 + 8" button is present.

At the bottom, there are three buttons: "GENERATE ⌘ + ⌘", "EXPLORE CTRL + SPACE", and "SHARE..."

Klik Generated dan buka folder yang sudah di extract pada IDE kita

## Springboot - Restful Web Services/REST Template - Sesi 17

# MySQL Database Configurations

Buat Database untuk store data user : Create, Update, Delete dan Retrieve semua Informasi User dengan REST API.

```
CREATE DATABASE DB;  
create table DB.users(  
id int NOT NULL AUTO_INCREMENT primary key,  
first_name varchar(50) NOT NULL,  
last_name varchar(50) NOT NULL);
```

Define connection properties in /src/main/resources/application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/DB  
spring.datasource.username=root  
spring.datasource.password=root1234
```



## Springboot - Restful Web Services/REST Template - Sesi 17

### Entity Model Class

Create the User model class to map with the Users table.

@Entity is used to annotate that the class is an entity in the database. @Table is used to annotate the name of the table in the database. @GeneratedValue is used to generate strategies for the values of primary keys.

```
package com.example.demo.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Table(name = "users")
public class User {
    private int id;
    private String firstName;
    private String lastName;

    public User() {
    }

    public User(int id, String firstName, String lastName) {
        this.id = id;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    public int getId() {
        return id;
    }

    //other setters and getters
}
```



## Springboot - Restful Web Services/REST Template - Sesi 17

### JPA Data Repository

Create User Repository interface extending JPA Repository.

There are built-in methods for CRUD operations in JpaRepository, writing any SQL query is not needed.

```
package com.example.demo.repository;  
  
import com.example.demo.model.User;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface UserRepository extends JpaRepository<User, Integer>  
{  
}
```



## Springboot - Restful Web Services/REST Template - Sesi 17

### Service Class

Create a User service class to code the business logic and it acts as a middle layer between repository and controller class.

@Transactional used to annotate methods are executed in transactions.

```
package com.example.demo.service;

import com.example.demo.model.User;
import com.example.demo.repository.UserRepository;
import org.springframework.beans.factory.annotation.Autowired;

import java.util.List;
@Service
@Transactional
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public List<User> listAllUser() {
        return userRepository.findAll();
    }

    public void saveUser(User user) {
        userRepository.save(user);
    }

    public User getUser(Integer id) {
        return userRepository.findById(id).get();
    }

    public void deleteUser(Integer id) {
        userRepository.deleteById(id);
    }
}
```



## Springboot - Restful Web Services/REST Template - Sesi 17

# Create REST Controllers class

Create Rest User Controllers class which contains all REST API endpoints for CRUD operations.

```
package com.example.demo.controller;

import com.example.demo.model.User;
import com.example.demo.service.UserService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.NoSuchElementException;

@RestController
@RequestMapping("/users")
public class UserController {
    @Autowired
    UserService userService;

    @GetMapping("")
    public List<User> list() {
        return userService.listAllUser();
    }

    @GetMapping("/{id}")
    public ResponseEntity<User> get(@PathVariable Integer id) {
        try {
            User user = userService.getUser(id);
            return new ResponseEntity<User>(user, HttpStatus.OK);
        } catch (NoSuchElementException e) {
            return new ResponseEntity<User>(HttpStatus.NOT_FOUND);
        }
    }

    @PostMapping("/")
    public void add(@RequestBody User user) {
        userService.saveUser(user);
    }
}
```



```
    @PutMapping("/{id}")
    public ResponseEntity<?> update(@RequestBody User user,
    @PathVariable Integer id) {
        try {
            User existUser = userService.getUser(id);
            user.setId(id);
            userService.saveUser(user);
            return new ResponseEntity<>(HttpStatus.OK);
        } catch (NoSuchElementException e) {
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }
    }
    @DeleteMapping("/{id}")
    public void delete(@PathVariable Integer id) {

        userService.deleteUser(id);
    }
}
```



## Springboot - Restful Web Services/REST Template - Sesi 17

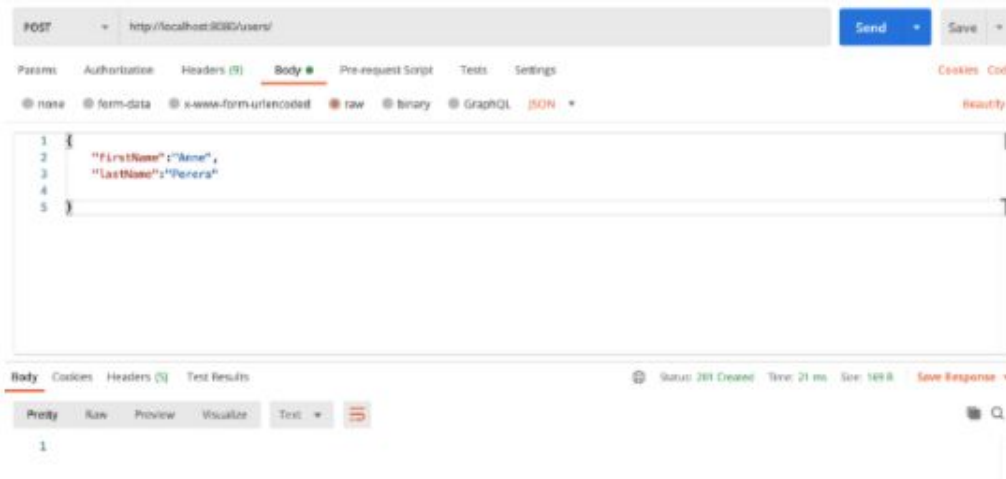
# Build & Run Projects

The app will start running at <http://localhost:8080>.

```
mvn spring-boot:run
```

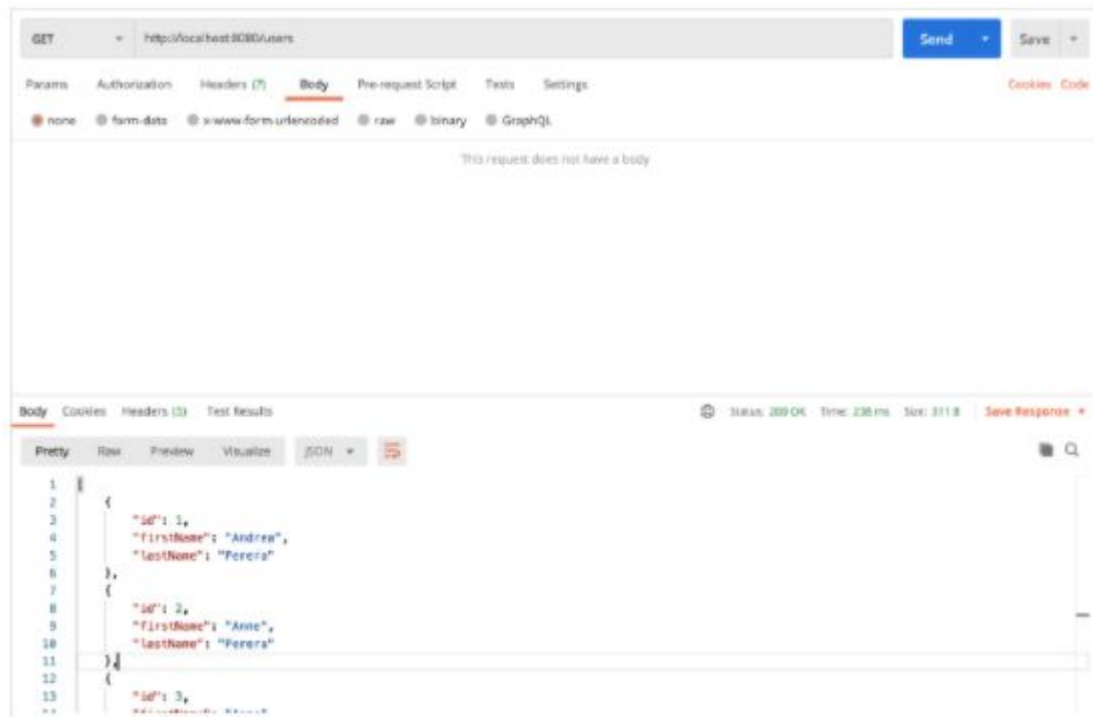
Testing using Postman

A. Execute POST Request method to add a user to the Database.





Execute the GET Request method to get all users from the Database.



Execute GET Request method along with id to get specific user details from the Database.

The screenshot displays a REST client interface with the following details:

- URL:** `http://localhost:8080/users/1`
- Method:** `GET`
- Body:** This request does not have a body.
- Response:**
  - Status:** 200 OK
  - Time:** 32 ms
  - Size:** 213 B
  - Save Response:** +
- Response Body (JSON):**

```
1 {
2   "id": 1,
3   "firstName": "Andrea",
4   "lastName": "Pereira"
5 }
```