



Lesson 33

08.04.2024

```
public class Test1 {  
    public static void main(String[] args) {  
        Test1 test = new Test1();  
        System.out.println(test == this);  
    }  
}
```

```
public class Test2 {  
    public static void main(String[] args) {  
        Test2 test = new Test2();  
        test.print("C");  
    }  
    public void print(){  
        System.out.println("A");  
    }  
    public static void print(String s){  
        System.out.println("B");  
    }  
}
```



```
public class Test3 {  
    public static void main(String[] args) {  
        Set<Number> set = new HashSet<>();  
        set.add(1);  
        set.add(1L);  
        set.add(1.0);  
        System.out.println(set.size());  
    }  
}
```

```
class MyLink{
    public MyLink(){
        str = "New";
    }
    public String str;
}

public class Test4{
    public static void main(String[] args) {
        MyLink b1 = new MyLink();
        MyLink b2 = b1;
        b2.str = "MyString";
        System.out.println(b1.str);

        String a1 = "Test";
        String a2 = a1;
        System.out.println(a2);
        a1 = "Not a Test";
        System.out.println(a2);
    }
}
```



5 java questions



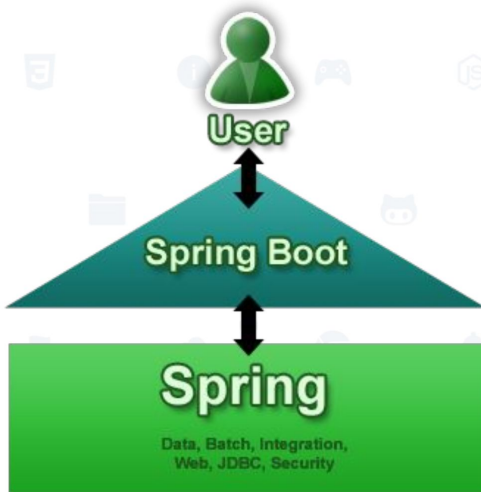
1. Як організовано HashMap
2. Чим відрізняються LinkedList та ArrayList
3. SQL: where vs having
4. Time / Space complexity
5. Проміжні та термінальні операції в Stream API



Що таке Spring Boot

Spring Boot – це доповнення до Spring, яке полегшує та прискорює роботу з ним. Сам Spring Boot представляє собою набір утиліт, що автоматизують налаштування фреймворку. Ось що він бере на себе:

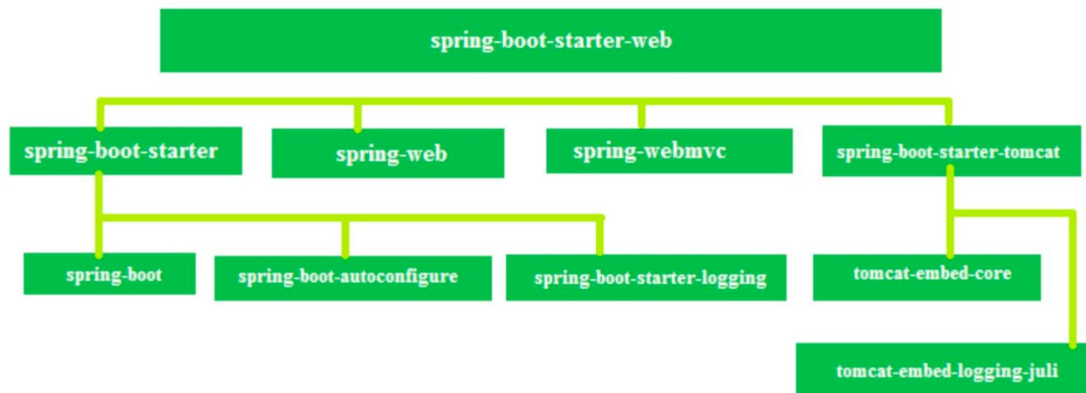
- упаковує залежності у starter-пакети;
- автоматично конфігурує програми за допомогою jar-залежностей;
- створює веб-сервер, що дозволяє локально запускати на ньому програми.






Простота управління залежностями

Щоб прискорити процес управління залежностями, Spring Boot неявно пакує необхідні сторонні залежності для кожного типу програми на основі Spring і надає їх розробнику за допомогою так званих starter-пакетів (spring-boot-starter-web, spring-boot-starter-data-jpa і т.д. д.).

Starter-пакети є набором зручних дескрипторів залежностей, які можна включити у свою програму. Це дозволить отримати універсальне рішення для всіх, пов'язаних зі Spring технологій, позбавляючи програміста від зайвого пошуку прикладів коду та завантаження з них необхідних дескрипторів залежностей (приклад таких дескрипторів та стартових пакетів буде показано нижче)



- 
- 
- `spring-boot-starter-web-services`: For building applications exposing SOAP web services
 - `spring-boot-starter-web`: Build web applications and RESTful applications
 - `spring-boot-starter-test`: Write great unit and integration tests
 - `spring-boot-starter-jdbc`: Traditional JDBC applications
 - `spring-boot-starter-hateoas`: Make your services more RESTful by adding HATEOAS features
 - `spring-boot-starter-security`: Authentication and authorization using Spring Security
 - `spring-boot-starter-data-jpa`: Spring Data JPA with Hibernate
 - `spring-boot-starter-cache`: Enabling the Spring Framework's caching support
 - `spring-boot-starter-data-rest`: Expose simple REST services using Spring Data REST
- 



Автоматична конфігурація

Другою чудовою можливістю **Spring Boot** є автоматична конфігурація програми.

Після вибору відповідного **starter-пакету**, Spring Boot спробує автоматично налаштувати Spring додаток на основі доданих вами jar-залежностей.

Наприклад, якщо ви додаєте **Spring-boot-starter-web**, Spring Boot автоматично конфігурує такі зареєстровані біни, як **DispatcherServlet**, **ResourceHandlers**, **MessageSource**.

Якщо ви використовуєте **spring-boot-starter-jdbc**, Spring Boot автоматично реєструє біни **DataSource**, **EntityManagerFactory**, **TransactionManager** та зчитує інформацію для підключення до бази даних із файлу `application.properties`.

Автоматична конфігурація може бути повністю перевизначена в будь-який момент за допомогою налаштувань користувача.


Вбудована підтримка сервера додатків - контейнера сервлетів

Кожен Spring Boot web-додаток включає вбудований web-сервер.

Розробникам тепер не треба турбуватися про налаштування контейнера сервлетів та розгортання програми на ньому. Тепер програма може запускатися сама, як виконуваний jar-файл з використанням вбудованого сервера.

Якщо вам потрібно використовувати окремий HTTP-сервер, для цього достатньо виключити залежність від замовчуванням. Spring Boot надає окремі starter-пакети для різних HTTP-серверів.

Створення автономних web-додатків із вбудованими серверами не тільки зручне для розробки, а й є допустимим рішенням для додатків корпоративного рівня і стає все більш корисним у світі мікросервісів. Можливість швидко запакувати весь сервіс (наприклад, аутентифікацію користувача) в автономному артефакті, що повністю розгортається, який також надає API — робить установку і розгортання програми значно простіше.



@SpringBootApplication : Містить інструкції `@ComponentScan`, `@Configuration` і `@EnableAutoConfiguration`. Серед них `@ComponentScan` дозволяє Spring Boot сканувати клас `Configuration` та додавати його до контексту програми.

@Configuration Еквівалентно XML-файлу конфігурації Spring; використовуйте код Java для перевірки типів безпеки.

@EnableAutoConfiguration Автоматичне налаштування.

@ComponentScan Сканування компонентів може автоматично виявляти та збирати деякі компоненти.

@Component Його можна використовувати з `CommandLineRunner` для виконання деяких основних завдань після запуску програми.

@RestController Анотація є набором `@Controller` і `@ResponseBody`, що означає, що це bean-компонент контролера, а значення функції, що повертається, безпосередньо заповнюється в тілі відповіді HTTP, який є контролером в стилі REST.

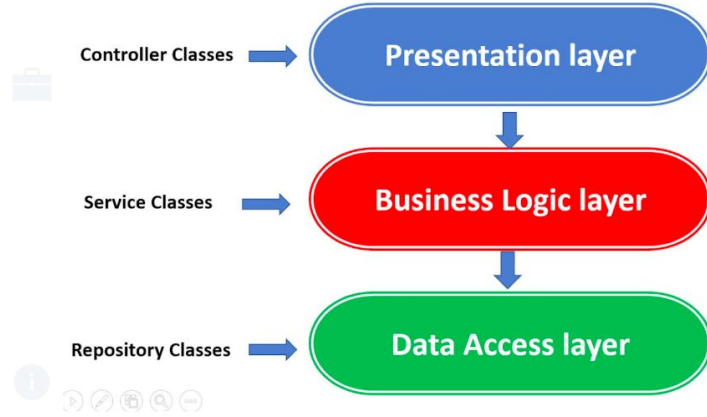
@Autowired Імпортувати автоматично.

@PathVariable Отримати параметри.

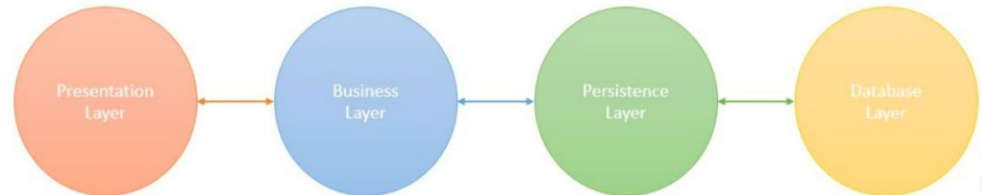
@RepositoryRestResource public Використовуйте з `spring-boot-starter-data-rest`.

@ControllerAdvice : Містять `@Component`. Можна сканувати. Одностаино обробляйте винятки.

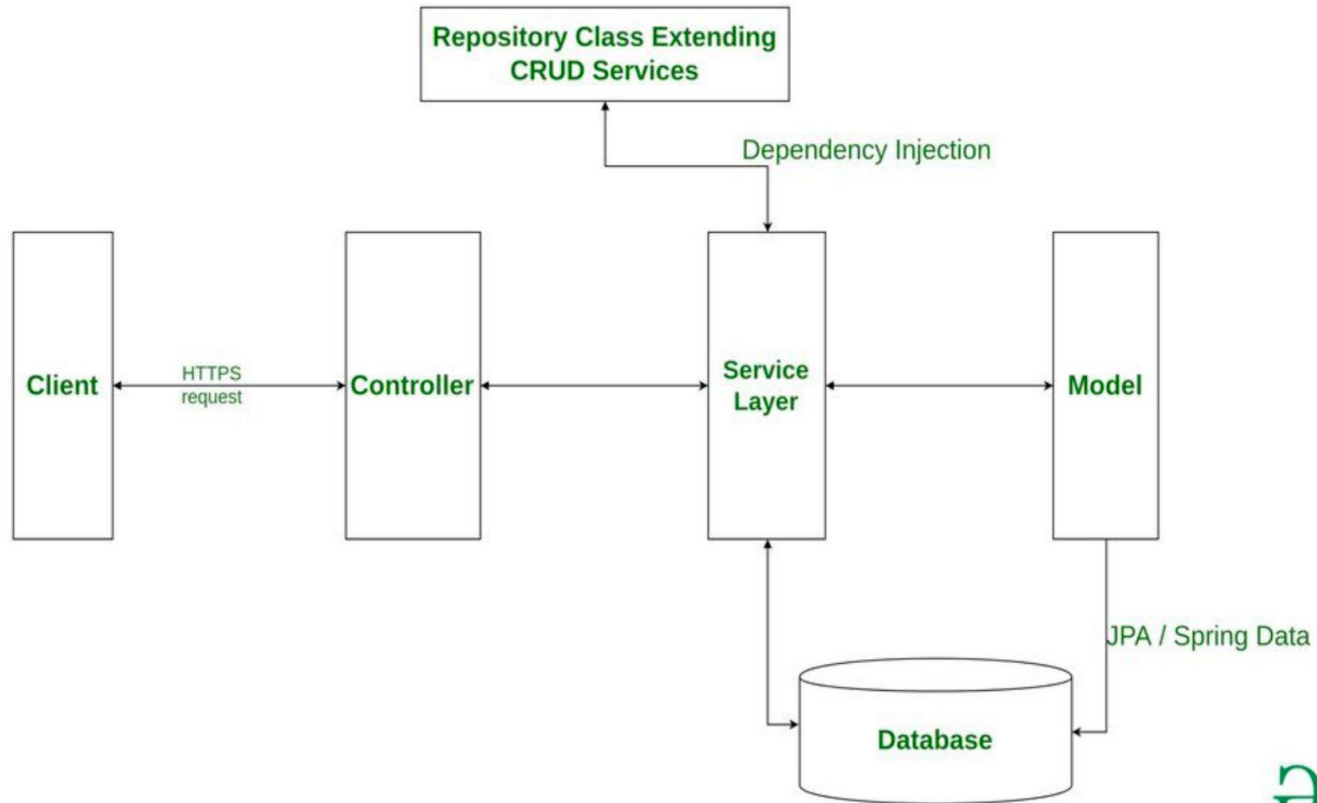
Three-Tier (or Three-Layer) Architecture in Spring MVC



1. **Presentation Layer** – Authentication & Json Translation
2. **Business Layer** – Business Logic, Validation & Authorization
3. **Persistence Layer** – Storage Logic
4. **Database Layer** – Actual Database



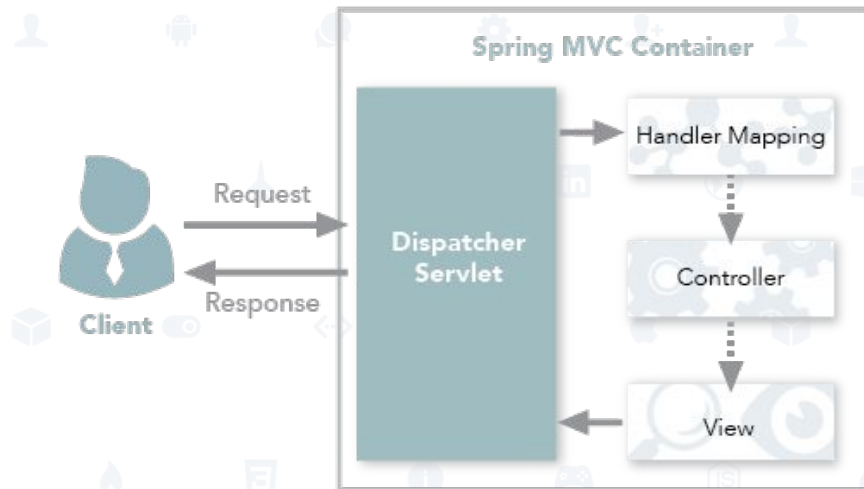
Spring Boot flow architecture



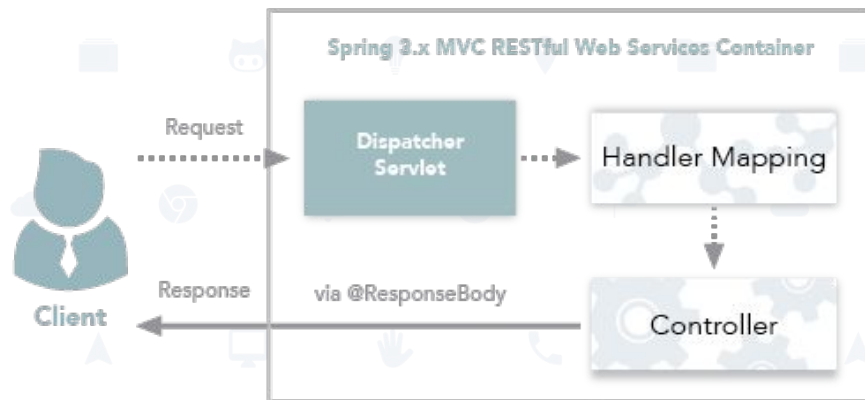
Explanation:

- The Client makes an **HTTP** request(GET, PUT, POST, etc.)
- The HTTP request is forwarded to the **Controller**. The controller maps the request. It processes the handles and calls the server logic.
- The business logic is performed in the **Service layer**. The spring boot performs all the logic over the data of the database which is mapped to the spring boot model class through **JPA** .
- The [JSP](#) page is returned as Response from the controller.

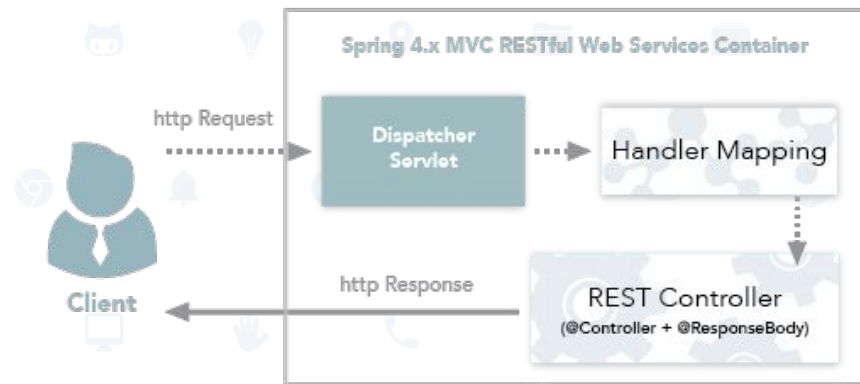
@Controller → HTTP response



@Controller + @ResponseBody



@RestController



HTTP/1.1 200 OK

Status Line

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache/2.2.14 (Win32)

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

Content-Length: 88

Content-Type: text/html

Connection: Closed

Headers

<html>

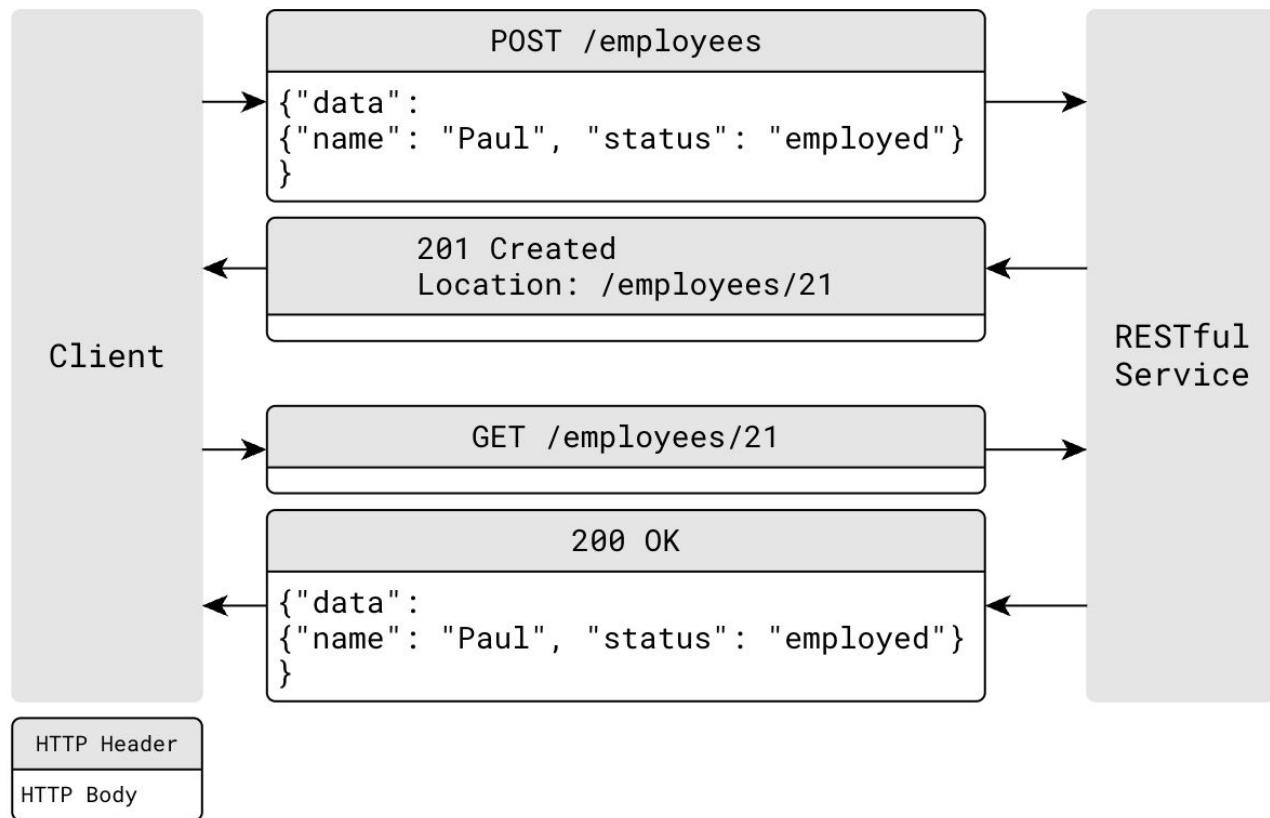
<body>

<h1>Hello, World!</h1>

</body>

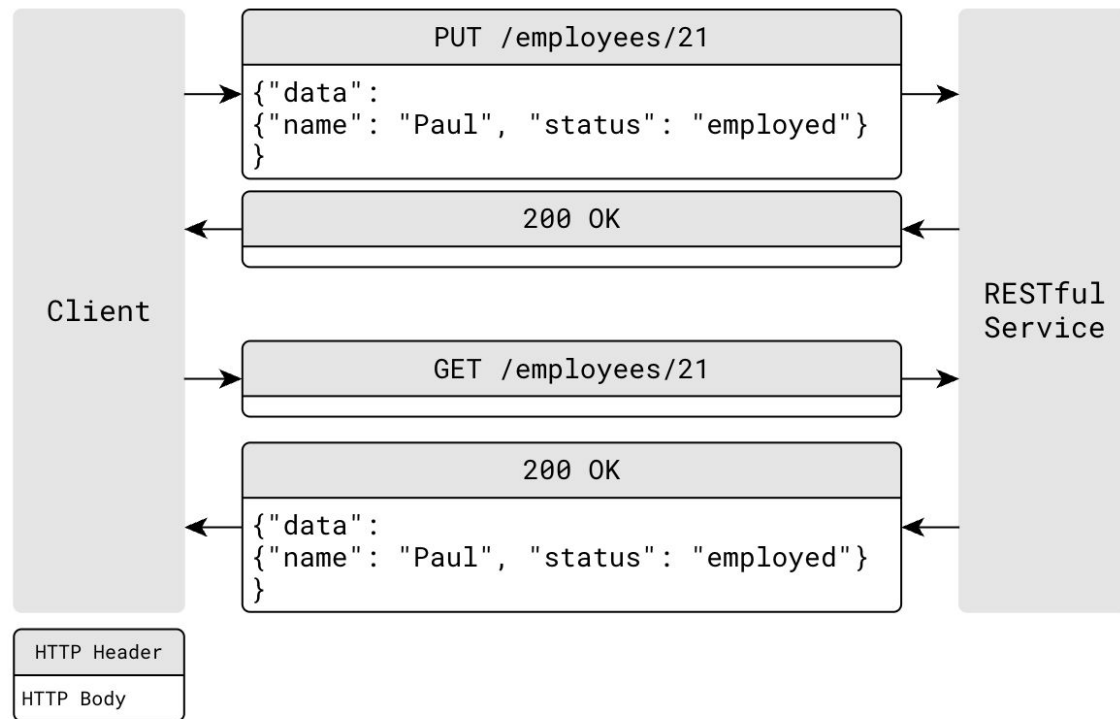
</html>

Body

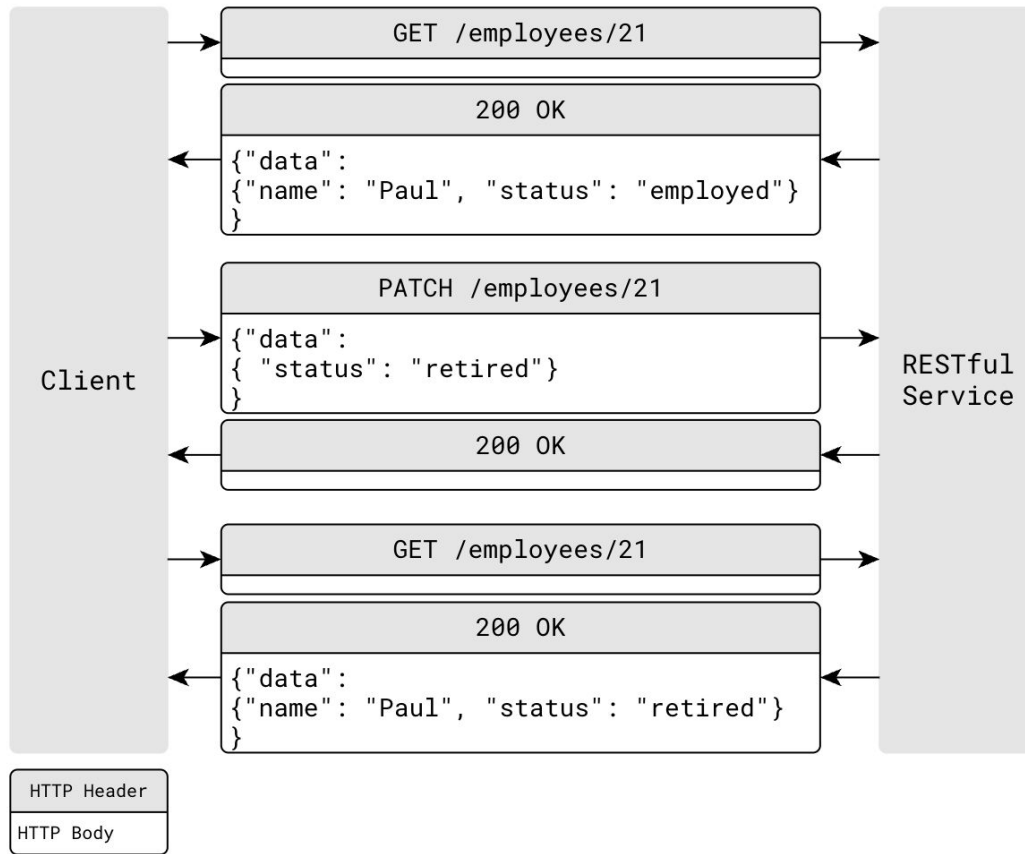


Use POST for creating a new resource

PUT on the Single Resource URL for Updating a Resource



Use PUT for updating an existing resource.



Use PATCH and send only the fields you like to update.

@PathVariable and @RequestParam and @RequestBody

http://localhost:8080/users/**101**/accounts?type=**current**&status=**active**

Path Variable

Request Parameter

Request Parameter

```
@GetMapping("/user/{id}/accounts")  
public List<Account> getAccounts(@PathVariable String id, @RequestParam String type, @RequestParam String status) {  
    //...  
}
```