REST SERVİSLER

REST (Representational State Transfer), temsili durum transferi kısaltması anlamında kullanılır. Web servis oluşturmak için kullanılan yöntemler arasındadır. Modern yapısı ve ekstra yük getirmeyen mimarisi ile sıkça kullanılan basit servislerdir. Yine sıkça kullanılan RESTFUL terimi ile karıştırılmaması gerekir. RESTFUL, REST mimarisi ile hazırlanmış web servislerine verilen isimdir.

REST, HTTP metotları kullanılarak çağrılır ve bu çağrılar içinde istenilen verilerin transferi gerçekleşir. Esnek ve kolay geliştirilebilir yapıları sebebiyle REST servisler tercih edilir.

Kullanılan HTTP metotları şöyle sıralanır;

GET: Verileri almak için kullanılır. Örnek olarak ben şuan için bir football-api ile çalışıyorum. Bunun için bana ülke, lig, takım gibi bilgiler gerekiyor ve benim bunları kullanıcıya göstermem gerekiyor. Ben kendi uygulamamda bu verilere erişmek ve bu verileri göstermek için GET metodu kullanıyorum. Mesela aşağıda bulunan "GET" metodu ile ben internette hazır olarak bulduğum bir servis içerisindeki tüm ülke isimlerine erişebiliyorum. Aşağıda bulunan ekran görüntüsü Kotlin ile geliştirdiğim bir projeden alınmıştır.

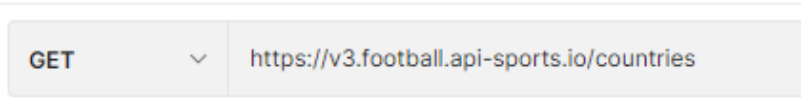
```
//Countries --> https://v3.football.api-sports.io/countries
@Streaming
@GET( value: "countries")
fun allCountries(@Header( value: "x-rapidapi-key")api_key : String, @Header( value: "x-rapidapi-host") host_name:String ) : Call<CountryResponse>
```

PUT: Veriler üzerinde ekleme, çıkartma yani kısacası güncelleme yapmamızı sağlayan metottur. Böylece veriler üzerinde istediğimiz değişikliği kolay bir şekilde yapabiliriz. GET metodu ile gelen ülkelerin özellikleri arasında istediğimiz değişikliği yapmamızı sağlar.

DELETE: Verileri silmek için kullandığımız metottur. İstenmediğimiz veya artık lazım olmayan verileri silmemizi sağlar.

POST: Yeni veri eklemek (göndermek) için kullandığımız metottur.

Görüldüğü üzere bu servisler basit yapıları ve kolay kullanımları ile geliştiricilerin işini kolaylaştırır. Yukarıda GET metodu ile gösterdiğim servisi POSTMAN ile birlikte test edebiliriz. Postman bize servislerimizi test etmemizi sağlar.



GET isteğimizi bu şekilde atabiliriz.

Çıktı yukarıdaki şekilde geliyor. Böylece kullandığımız servis metodunu test etmiş olduk.

Çıktı formatı JSON formatındadır. Yaygın kullanım olarak, REST servislerin çıktısı JSON formatında olur.

REST SERVİSLER NASIL YAPILIR ?

Java, Spring Boot frameworkü ile birlikte RESTFUL web servis geliştirileceğini açıklayacağız. Bunun için öncelikle Spring Boot projemizi oluşturup başlatırız. Sonrasında herhangi bir veri tabanı ile bağlantı sağlarız. Bu MySQL, PostgreSQL vs. olabilir. Spring Boot projemiz için gerekli birkaç dependency eklememiz gerekir. Bunlar Web, DevTools, Spring Data JPA ve kullanacağımız veri tabanına ait olan veri tabanı dependency özelliği olacaktır. Veri tabanı bağlantısını, application.properties üzerinden gerekli özellikler ile birlikte yönetebiliriz.

Projeye ilk olarak bir model sınıfı oluştururuz. Bu model bizim projemizde gerekli olacak nesnelerin isimlerine göre oluşturulabilir. Örnek vermek gerekecek olursa ben yukarıda bir football-api üzerinden GET metodunu gösterdim. O örnekte ülkeler için gerekli olacak bilgiler ile ilgili bir model sınıfı oluşturabiliriz. Oluşturulacak olan sınıf @Entity anotasyonu olarak işaretlenmelidir. Bu anotasyon Hibernate ile birlikte sınıfımızın, veri tabanı tarafında tablo oluşturacağı anlamına gelir.

CRUD işlemler için, yani create(oluşturma), read(okuma), update(güncelleme) ve delete(silme) işlemleri kullanmak için Repostiyory interface sınıfı oluşturulur. Bu sınıf @Repository anotasyonu ile işaretlenir ve JpaRepository üzerinden miras alır. Yani oluşturduğumuz interface extends eder. Repository interfaceimizi, servis sınıfı içerisinde Autowired ederek gerekli hazır fonksiyonlara ulaşırız.

Servis sınıfı yazarak bu sınıfta, Controller tarafından kullanmak istediğimiz metotları ihtiyaçlarımız doğrultusunda yazarız. Servis katmanında yazdıklarımızı oluşturacağımız Controller ile kullanacağız.

Sonrasında artık ihtiyacımız olan bir Controller sınıfı çıkar. Bu sınıf içerisinde yukarıda bahsedilen GET, POST, DELETE gibi metotlar tanımlanır. Bu sınıf @RestController anotasyonu ile işaretlenmesi gerekir. Java tarafında bu istekler @GetMapping, @PostMapping gibi isimlerle metotlar üzerinde işaretlenerek kullanılır.

Spring projemizi çalıştırdığımız zaman bu yaptığımız örneği Postman aracılığı ile veya localhost üzerinde ayarladığımız port üzerinden gözlemleyebiliriz. Artık bizimde elimizde bir REST servisimiz oluşmuş oldu.

Şimdi sırayla bu anlatılanları uygulayabiliriz. Herhangi bir veri tabanı ile bağlantı sağlamayacağım için @Entity anotasyonunu kullanmayacağım.

Model Sınıfımız

```
@Data
@NoArgsConstructor
@AllArgsConstructor

public class Player {

    private int id;
    private String name;
    private String lastName;
    private String teamName;
    private int age;

}
```

Repository interface

```
public interface PlayerRepository extends JpaRepository<Player, Integer> {


}
```

Servis sınıfımız

```java
@Service

public class PlayerService implements PlayerControlService {


    @Autowired
    private PlayerRepository playerRepository;


    @Override
    public Player createPlayer(Player player) {
        return playerRepository.save(player);

    }

    @Override
    public Player updatePlayer(Player player,int playerId) {
        Player updatePlayer = playerRepository.findById(playerId).get();
        updatePlayer.setAge(player.getAge());
        updatePlayer.setTeamName(player.getTeamName());
        return playerRepository.save(updatePlayer);

    }

    @Override
    public Player deletePlayer(int playerId) {
        Player player = playerRepository.findById(playerId).get();
        playerRepository.deleteById(playerId);
        return player;

    }

    @Override
    public List<Player> getAllPlayer() {

        return this.playerRepository.findAll();
    }



}
```

Controller Sınıfımız

```java
@RestController
public class PlayerController {


    @Autowired
    private PlayerControlService playerService;


    @GetMapping(value = "/players")
    public List<Player> getAllPlayer(){
        List<Player> playerList = playerService.getAllPlayer();
        return playerList;

    }

    @PostMapping(value="/players")
    public Player createPlayer(@RequestBody Player player) {
        return playerService.createPlayer(player);

    }


    @PutMapping(value = "/player")
    public Player updatePlayer(@RequestBody Player player) {
        return playerService.updatePlayer(player,player.getId());

    }

    @DeleteMapping(value="/player/{id}")
    public void deletePlayer (@PathVariable int id) {
        playerService.deletePlayer(id);
    }
}
```