# 1 - SPRING INTRO (BA 19/06/2023)

**Roadmap**

- Spring Boot
- Hibernate (JPA)
- Spring Modules
- Web App
- Rest API
- MVC
- React
- Unit Test
- Integration Test
- Docker
- Cloud

## Neden Spring?

Java programlamayı daha hızlı, daha basit ve herkes için daha güvenli yapar. Spring'in hız, basitlik ve verimlilik odaklı olması onu dünyanın en popüler Java frameworkü haline getirmiştir.

(*Spring is flexible*) At it's core, Spring Framework's ***Inversion of Control (IoC)*** and ***Dependency Injection (DI)*** features provide the foundation for a wide-ranging set of features and functionality.

Spring projects also support the ***reactive (nonblocking) programming model*** for even greater efficiency.

## Tomcat Server

Server'a neden ihtiyacımız olur?

Web teknolojilerinde bir client, bir de server oluyordu. Client, Server'a istekte bulunuyor. Server içerisinde işletim sistemi, eğer bir web sunucusu olacaksa web sunucusu özelliğine sahip bir yazılım var. Spring'de Tomcat embedded olarak var. Bu yapı sayesinde bir sürü ayarla vs. ile uğraşmamış oluyoruz ve şaşırtıcı derecede hızlı bir yapı sunmuş oluyor. Normalde herhangi bir dilde, örneğin PHP'de .NET'te bir program yazarken mutlaka bir sunucu olması lazım, o sunucuya bunların yüklenmesi lazım, ayarların yapılması lazım. Ama burada Tomcat ayarlanmış bir şekilde bulunduğundan uygulama direkt ayağa kalkabiliyor.

# API

API'ler, iki yazılım bileşiminin belirli tanımlar ve protokoller aracılığıyla birbirleriyle iletişim kurmasına olanak sağlayan mekanizmalardır. Örneğin, meteoroloji müdürlüğünün yazılım sistemi, günlük hava durumu verilerini içerir. Telefonlardaki hava durumu uygulaması, API'ler aracılığıyla bu sistemle konuşur ve telefonda günlük hava durumu güncellemelerini gösterir.

**API Örnekleri**

İş Bankası API Portal

Binance APIs

## REST API

REST, Temsili Durum Aktarımı anlamına gelen Representational State Transfer ifadesinin kısaltmasıdır. REST, istemcilerin sunucu verilerine erişirken kullanabilmesi için GET, PUT, DELETE gibi belirli işlevler kullanır. İstemciler ve sunucular, HTTP üzerinden veri alışverişi yapar.

REST API'sinin ana özelliği durumsuz olmasıdır. Durumsuz olması, sunucuların istekler arasında istemci verilerini kaydetmemesi anlamına gelir. İstemcinin sunucuya gönderdiği istekler, bir web sitesini ziyaret etmek için tarayıcınıza yazdığınız URL'lere benzer. Gelen yanıt ise bir web sayfasında görmeye alışık olduğumuz grafikleri içermeyen sade verilerdir.

REST API'lerinin sunduğu dört temel avantaj vardır:

**Entegrasyon**

API'ler, yeni uygulamaların mevcut yazılım sistemlerine entegre edilmesi için kullanılır. Her bir işlevi sıfırdan yazmaya gerek duyulmadığından geliştirme hızı artar. Mevcut kodlardan yararlanmaya devam etmek için API'leri kullanabilirsiniz.

**İnovasyon**

Yeni bir uygulamanın gelişi, bütün bir sektörü değişime zorlayabilir. İşletmelerin hızlı yanıt vermesi ve yenilikçi hizmetlerin hızlı dağıtımını desteklemesi gerekir. Bunu, tüm kodu yeniden yazmak yerine API düzeyinde değişiklikler yaparak gerçekleştirebilirler.

**Genişleme**

API'ler, müşterilerinin ihtiyaçlarını birden fazla platformda karşılamak isteyen işletmelere benzersiz bir fırsat sunar. Örneğin, haritalar API'si harita bilgileri entegrasyonunun web siteleri, Android, iOS vb. aracılığıyla yapılabilmesini sağlar. Her işletme, ücretsiz veya ücretli API'leri kullanarak kendi dahili veritabanlarına benzer şekilde erişim verebilir.

**Bakım kolaylığı**

API, iki sistem arasında bir ağ geçidi vazifesi görür. Her bir sistem, API'nin etkilenmemesi için dahili değişiklikler yapma gereği duyar. Bu sayede, taraflardan birinin gelecekte yapacağı kod değişiklikleri, diğer tarafı etkilemez.

# EXAMPLE PROJECT

## Spring Initializr

https://start.spring.io/:

- Project: Gradle-Groovy
- Language: Java
- Spring Boot: Last one (2.7.14)
- Project Metadata
    - Artifact: Project Name
    - Package name: org.hulyam

### Dependencies

- Lombok
- Spring Boot Dev Tools
- Spring Web
- Spring Data JPA
- PostgreSQL Driver


## application.yml & application.properties

Proje ile alakalı genel ayarların tutulduğu dosyalardır.

.properties veya YAML (.yml) dosyası olabilir. YAML dosyası olması yazımı kolaylaştırır. Daha sonra cloudla çalışırken yaml üzerinden gideceğimiz için şimdiden yaml dosyası oluşturuyoruz.

Öncelikle parametre ardından iki nokta üst üste ve **bir boşluk** bırakıldıktan sonra değer girilik. ***Eğer boşluk bırakılmazsa app hata verecektir.***

Alt kırılımlarda da **bir TAB** boşluk olmalıdır.

**application.yml:**

```
server:
  port: 9090

spring:
  datasource:
    driver-class-name: org.postgresql.Driver
    username: postgres
    password: root
    url: jdbc:postgresql://localhost:5432/DbSpringIntro
  jpa:
    hibernate:
      ddl-auto: create-drop
```

**application.properties:**

```
server.port=9090

spring.datasource.driver-class-name=org.postgresql.Driver
spring.datasource.username=postgres
spring.datasource.password=root
spring.datasource.url=jdbc:postgresql://localhost:5432/DbSpringIntro

spring.jpa.hibernate.ddl-auto=create-drop
```

## Layers and Annotations

> ***NOT:*** *Projeyi https://start.spring.io/ üzerinden oluşturduğumuz için Main (SpringExercies1IntroApplication) class'a @SpringBootApplication anotasyonunu ve "SpringApplication.run..." kodunu otomatik olarak ekliyor.*
>
> ***Main.class****:*

```
package org.hulyam;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class SpringExercies1IntroApplication {

 public static void main(String[] args) {
     SpringApplication.run(SpringExercies1IntroApplication.class,
args);
 }

}
```

## Repository

### Entity

Burada her zaman eklediğimiz Lombok Entity anotasyonlarını "User" örnek clasına ekliyoruz.

**User.class**:

```
package org.hulyam.repository.entity;

import lombok.AllArgsConstructor;
import lombok.Builder;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;

@Builder
@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
@Table(name = "tbluser")
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    Long id;
    String name;
```

```
    String address;
}
```

### IUserRepository Interface

- **Spring Annotation:** @Repository
- extends JpaRepository<User, Long>

```
package org.hulyam.repository;

import org.hulyam.repository.entity.User;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface IUserRepository extends JpaRepository<User, Long> {
}
```

## Service

- **Spring Annotation:** @Service
- **Lombok Annotation:** @RequiredArgsConstructor

```
package org.hulyam.service;

import lombok.RequiredArgsConstructor;
import org.hulyam.repository.IUserRepository;
import org.hulyam.repository.entity.User;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
@RequiredArgsConstructor
public class UserService {
    private final IUserRepository userRepository;

    public User save(User user) {
        return userRepository.save(user);
    }

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

```
}
```

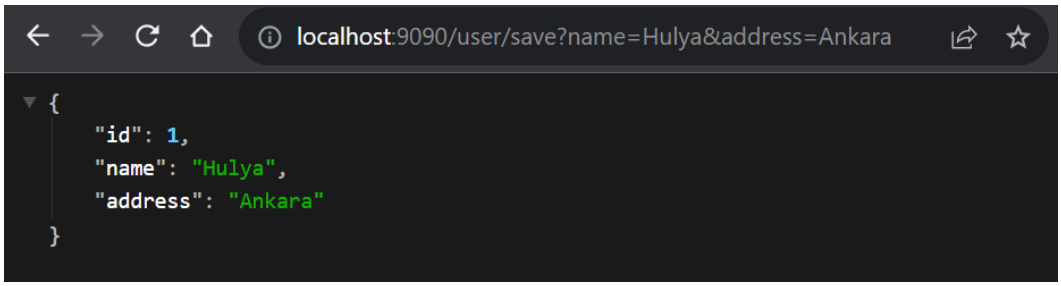
## Controller

- **Spring Annotation:** @RestController
- **Spring Annotation:** @RequestMapping("/user")
- **Lombok Annotation:** @RequiredArgsConstructor

```
package org.hulyam.controller;

import lombok.RequiredArgsConstructor;
import org.hulyam.repository.entity.User;
import org.hulyam.service.UserService;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
@RequestMapping("/user")
@RequiredArgsConstructor
public class UserController {
    private final UserService userService;



    @GetMapping("/save")
    public ResponseEntity<User> save(String name, String address) {
        return ResponseEntity.ok(userService.save(
                User.builder().name(name).address(address).build()
        ));
    }

    @GetMapping("/findall")
    public ResponseEntity<List<User>> findAll() {
        return ResponseEntity.ok(userService.findAll());
    }
}
```

> ***@GetMapping(/save)*** *UserController'da yazdığımız save metodunun "//localhost:9090/user/save" bağlantısı ile çağırılabileceğini belirtir.*

> ***Metoda bağlantı üzerinden parametre gönderme***
>
> *//localhost:9090/user/save?name=Hulya&address=Ankara*
>
> *Yukarıdaki bağlantıya gidildiğinde Controller'daki save metodundan dönen JSON objesi:*




## ResponseEntity

Controller'da returnler artık ResponseEntity olarak girilir. Kullanıcıya dönülecek olan datanın Entity olarak dönmesini ve bunun JSON formatında dönmesini sağlamak için kullanılır.

ResponseEntity sayesinde Controller'daki metotlar geriye JSON objesi dönerler. Kullanılma sebebi de budur.

### Neden JSON?

JSON alternatifi XML veri dönüşüdür. XML eskiden daha çok kullanılıyordu ve dosya boyutu JSON'a göre daha fazladır. Bu nedenle JSON artık neredeyse standarta dönüşmüş durumdadır.

## Mapping Annotations

- @GetMapping
- @PostMapping
- @DeleteMapping
- @RequestMapping
- @PatchMapping
- @PutMapping

Gelen isteğin türüne göre uygun işaretlenmiş metotlarla yönlendirme gerçekleştirirler.

> *Tarayıcı üzerinden GET request dışında bir istekte bulunamayız. Deneysel olarak requestlerin incelenmesi için **Postman** ve **Swagger** gibi uygulamalar kullanılabilir.*

**Request Types:**

- GET
- POST
- PUT
- PATCH
- DELETE
- HEAD

# APPENDIX A - LINKS

| Name | Link |
|---|---|
| Spring | https://spring.io/ |
| Spring Initializr | https://start.spring.io/ |
| Spring Training and Certification (VMware) | https://spring.academy/paths/spring-certified-professional-2023 |
| Postman (REST API Test App) | https://www.postman.com/ |
| Swagger (REST API Documentation App) | https://swagger.io/ |

# APPENDIX B - SPRING ANNOTATIONS

Spring, anotasyonlar ile sistemi yönetir. Nesne yaratmak için belli anotasyonlar kullanır.

| Annotation | Description |
| --- | --- |
| @SpringBootApplication | Spring'in diğer anotasyonları fark edip projeyi ona före çalıştırmasını sağlayan asıl anotasyondur. Bu anotasyon sayesinde içinde bulunduğu package ve onun altındaki packageların içindeki tüm dosyalarda diğer anotasyonları arar ve bulduklarıyla gerekli işlemleri yapar. |
| @Repository | |
| @Service | |
| @RestController | MVC ile çalışırken "@Controller" anotasyonu kullanılır. REST API projesinde ise bu anotasyon kullanılır.<br>REST API'de "findall" gibi bir istek atıldığında geriye data döner. REST API'de yapı data transferi üzerine kuruludur. Bu nedenle çok daha hızlıdır. Bu nedenle bir çok kuruluş arka planda REST API kullanmakta veya bu yapıya geçmektedir. |
| @RequestMapping | Uygulamaya gelen isteklerin URL içinden analiz edilip ulaşması gereken sınıfa yönlendirilmesi için kullanılır. Bir istek geldiği zaman bu sınıf hangi isteği karşılayacak bunu belirttiğimiz anotasyondur. Aslında arka planda *filtreleme* işi yapar. |
| @GetMapping | GET isteğinde bulunulursa karşılayacak metot işaretlenir. |