



REpresentational State Transfer (REST) i Spring Boot

Uvod

- ▶ Definiše principe softverske arhitekture za web
- ▶ Definiše model kako bi web trebao da funkcioniše
- ▶ **Nije standard**, ali se oslanja na upotrebu nekoliko standarda
- ▶ Predstavlja stil softverske arhitekture koji se zasniva na postojanju resursa i uniformnog upravljanja njima putem skupa predefinisanih operacija
- ▶ *Stateless*

REST principi

▶ **Resurs**

- ▶ svaki entitet na web-u je resurs (HTML stranica, XML dokument, slika...)
- ▶ bilo šta što se može imenovati može biti resurs

▶ **Identifikator resursa**

- ▶ svaki resurs mora da ima svoj jedinstveni identifikator putem kojeg mu se može pristupiti
- ▶ REST koristi *Uniform Resource Identifier* (**URI**) koji je definisan u okviru RFC 2396

▶ **Reprezentacija resursa**

- ▶ resurs je odvojen od konkretnog formata reprezentacije
- ▶ isti resurs može biti predstavljen različitim formatima (HTML, XML, JSON...)

REST principi

▶ **Operacije nad resursima**

- ▶ nad resursima se obavljaju jednostavne operacije
- ▶ operacije nad resursima pripadaju predefinisanom skupu: *Create, Read, Update, Delete*

▶ **Protokol za komunikaciju**

- ▶ najčešći protokol za komunikaciju je **HTTP**
- ▶ operacijama nad resursima odgovaraju HTTP metode:
 - ▶ **GET** – za čitanje
 - ▶ **POST** – za kreiranje
 - ▶ **PUT** – za ažuriranje
 - ▶ **DELETE** – za brisanje

REST principi

▶ **Operacije nad resursima**

- ▶ nad resursima se obavljaju jednostavne operacije
- ▶ operacije nad resursima pripadaju predefinisanom skupu: *Create, Read, Update, Delete*

▶ **Protokol za komunikaciju**

- ▶ najčešći protokol za komunikaciju je **HTTP**
- ▶ operacijama nad resursima odgovaraju HTTP metode:
 - ▶ **GET** – za čitanje
 - ▶ **POST** – za kreiranje
 - ▶ **PUT** – za ažuriranje
 - ▶ **DELETE** – za brisanje

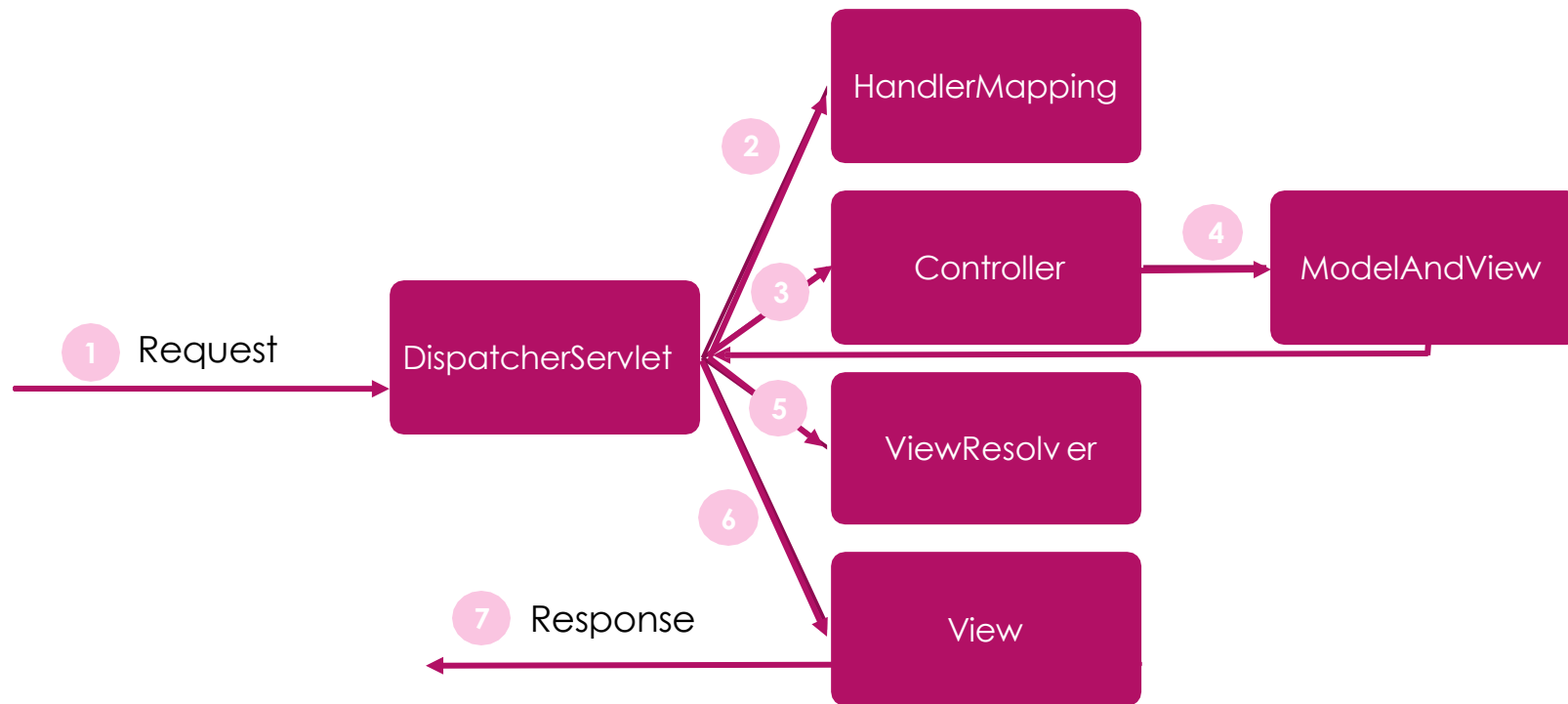
Spring i REST

- ▶ Spring podržava dva načina za implementaciju REST web servisa:
 1. Korišćenje MVC šablona
 2. Korišćenje konvertora HTTP poruka

Spring MVC

- ▶ Razdvaja aplikaciju na tri sloja:
 1. *Model*
 2. *View*
 3. *Controller*
- ▶ *view* se zasniva na *server-side scripting* tehnologijama (najčešće JSP)
 - ▶ ovo ne odgovara savremenim pristupima razvoja web aplikacija koji razdvajaju klijentsku i serversku aplikaciju
- ▶ *controller* sloj se može koristiti i ako je *view* realizovan na klijentu

Spring MVC obrada zahteva



Spring MVC obrada zahteva

- ▶ U Springu se MVC šablon razlikuje od standardnog po tome što se ispred kontrolera koji se pišu nalazi tzv. *Front Controller* koji prihvata sve zahteve korisnika i prosleđuje ih konkretnim kontrolerima
- ▶ Tačan naziv front kontrolera je `org.springframework.web.servlet.DispatcherServlet`
- ▶ *Handler Mapping* komponenta je zadužena za analizu zahteva na osnovu kojeg zaključuje koji tačno kontroler treba da obradi zahtev
 - ▶ moraju da se poklapaju i URL i tip metode

Spring MVC kontroler

- ▶ Kontroler je klasa anotirana sa `@Controller` ili `@RestController`
- ▶ Metode klase kontrolera su antoirane sa `@RequestMapping` anotacijom koja opisuje zahtev koji treba biti obrađen u toj metodi (URL i tip HTTP metode)
- ▶ `@RequestMapping` anotacija se može navesti i na nivou klase – tada sve metode unutar kontrolera imaju u svom URL-u prefiks koji je definisan u toj anotaciji
- ▶ Metode unutar istog kontrolera mogu biti definisane tako da imaju isti URL, ali se u tom slučaju moraju razlikovati po tipu HTTP metode koju obrađuju

Primer GET zahteva

► `@RequestMapping(value = "/api/greetings", method = RequestMethod.GET)`



URL koji određuje
putanju do metode



Tip HTTP metode

- Ukoliko se ne navede tip metode, podrazumeva se GET metoda
- Skraćeni zapis: `@GetMapping`

Preuzimanje parametara iz zahteva

► Parametre je u kontrolere moguće poslati na dva načina:

1. kao parametar koji je promenljiva u URL-u zahteva – *Path Variable*

```
@RequestMapping(value="api/greetings/{id}", method=RequestMethod.GET)
public void getGreeting(@PathVariable Long id) { ... }
```

Vrednost promenljive *id* se automatski popunjava na osnovu vrednosti iz URL-a. Ukoliko se naziv parametra metode i **promenljivog dela URL-a** ne poklapa, u okviru `@PathVariable` anotacije je potrebno navesti atribut *name* čija vrednost će da odgovara promenljivom delu URL-a (između vitičastih zagrada)

2. kao parametar HTTP zahteva – *Query Param*

```
public void getGreeting(@RequestParam Long id) { ... }
```

Spring parsira parametre HTTP zahteva. Potrebno je u metodi definisati parametar i anotirati ga odgovarajućom anotacijom. Podrazumeva se da naziv parametra funkcije odgovara nazivu parametra zahteva

REST u Spring MVC arhitekturi

- ▶ REST operacije GET, PUT, DELETE i POST se realizuju kroz kontrolere koji obrađuju istoimene HTTP metode
- ▶ Identifikacija resursa na koji se operacija odnosi se vrši korišćenjem parametrizovanih URL-ova i `@PathVariable` anotacije
- ▶ Format reprezentacije resursa se određuje korišćenjem odgovarajućeg konvertera podataka
- ▶ Konverteri omogućuju i pretvaranje podataka iz zahteva u Java objekte

Reprezentacija REST resursa

- ▶ Spring podržava dva načina za transformaciju REST resursa iz Java objekata u format koji klijent očekuje:

1. *Content negotiation*

- ▶ ovaj pristup koristi klasični Spring MVC mehanizam za kreiranje view objekta koji se šalje klijentu
- ▶ tretira resurs slično kao JSP stranicu koja se preko view komponente priprema, pa šalje klijentu
- ▶ preuzima se odgovarajući view koji vrši transformaciju resursa u ciljni format

2. **Konverzija poruke**

- ▶ poseban konvertor samo pretvara Java objekat u ciljni format bez prolaska kroz kompletan Spring MVC tok
- ▶ jednostavniji pristup

Thymeleaf

- ▶ <https://www.thymeleaf.org/doc/tutorials/3.0/thymeleafspring.html>
- ▶ Server side rendering engine



Konvertori HTTP poruka

- ▶ Konvertor se automatski primenjuje na osnovu formata resursa koji je metoda naznačila da vraća:

```
@RequestMapping(value = "/api/greetings",  
                method = RequestMethod.GET,  
                produces = MediaType.APPLICATION_JSON_VALUE)  
public @ResponseBody List<Greeting> getGreetings() { ... }
```

- ▶ Može se naznačiti da se konvertor poruka poziva automatski – u tom slučaju se kontroler mora anotirati sa `@RestController` anotacijom. Tada nema potrebe da se povratni tip metode anotira sa `@ResponseBody`

Konvertori HTTP poruka

- Spring podržava i drugi smer konverzije - konverzija podataka iz u tela HTTP zahteva:

```
@RequestMapping(value = "/api/greetings",  
                method = RequestMethod.POST,  
                consumes = MediaType.APPLICATION_JSON_VALUE,  
                produces = MediaType.APPLICATION_JSON_VALUE)  
public @ResponseBody Greeting createGreeting(@RequestBody Greeting greeting) { ... }
```

- Očekivani Java objekat je potrebno definisati kao parametar metode i taj parametar anotirati **@RequestBody** anotacijom. U zavisnosti od tipa podatka koji metoda očekuje (definiše se kao vrednost *consumes* atributa) će se pozvati odgovarajući konvertor

ResponseEntity

- ▶ Metode kontrolera mogu da vrate i *ResponseEntity* objekat koji sadrži više informacija o odgovoru
- ▶ *ResponseEntity* objekat može da sadrži:
 - ▶ **telo** (podatke) - metode anotirane sa `@ResponseBody` sadrže samo telo
 - ▶ **zaglavlje** (metapodatke)
 - ▶ HTTP **status** kod

Paginacija i sortiranje

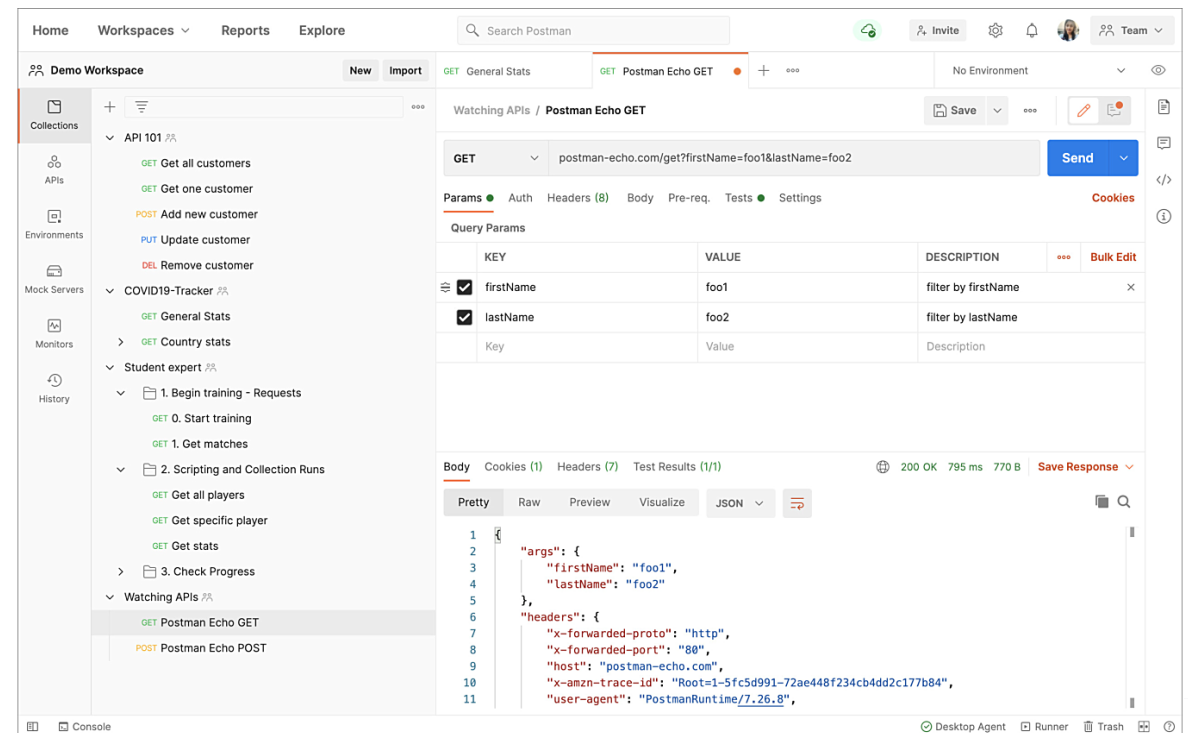
- ▶ Metoda može da kao parametar prima interfejs *Pageable*. U tom slučaju se automatski prepoznaju sledeći parametri URL-a:
 - ▶ **page** – broj stranice koja se dobavlja (prva stranica ima indeks 0)
 - ▶ **size** – broj entiteta po stranici
 - ▶ **sort** - kriterijum sortiranja koji se zadaje u formatu `fieldName, asc` ili `fieldName, desc`
- ▶ Primer URL-a:

`http://localhost:8080/api/greetings?sort=id,desc&page=1&size=3`

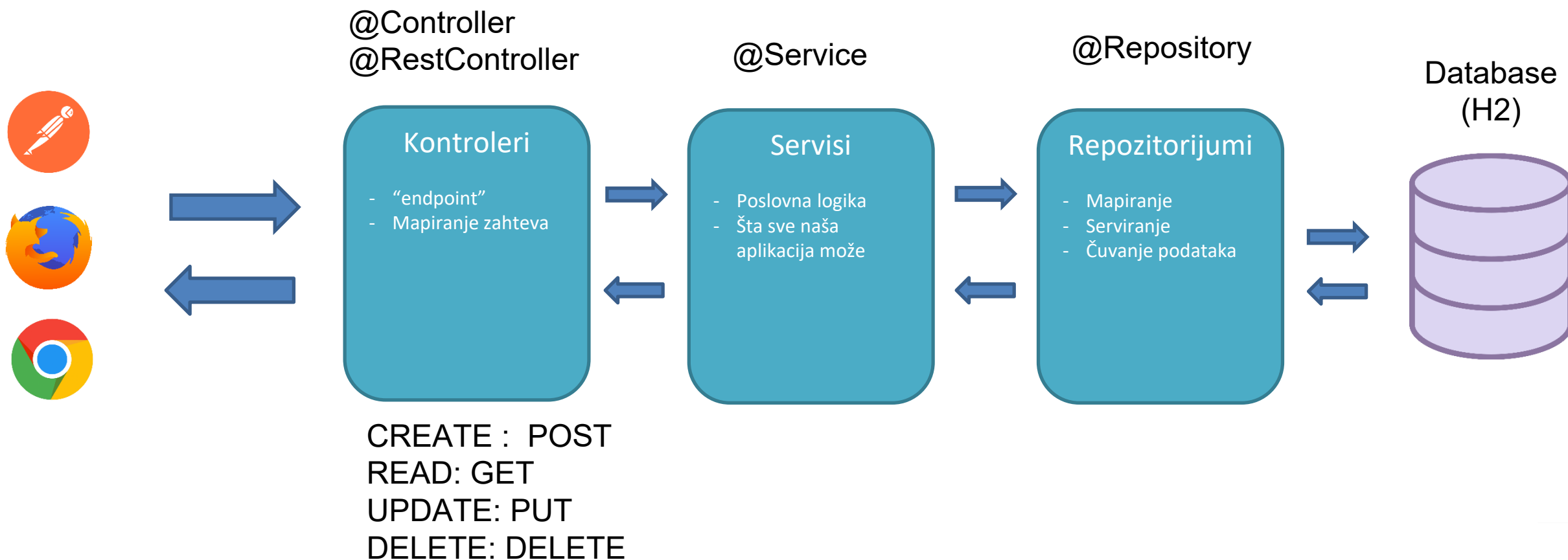


Postman

- ▶ Koristicemo ga za prikaz odgovora nasih REST kontorlera
- ▶ „Api za razvoj drugih api-ja“
- ▶ <https://www.postman.com/downloads/>



Standardna arhitektura



Inversion of Control (IoC)

- ▶ *Inversion of Control (IoC)* je tehnika kojom se povezivanje objekata vrši u toku izvršavanja (*runtime*) a ta veza nije poznata u toku kompiliranja (*compile time*)
- ▶ Da bi kontejner mogao da izvrši povezivanje, objekti moraju imati odgovarajuće uputstvo
- ▶ Jedna forma IoC je dependency injection (DI)

Inversion of Control (IoC)

- ▶ Spring kontejner je srž Spring radnog okvira
- ▶ Spring kontejner koristi *dependency injection* (DI) da upravlja komponentama koje čine aplikaciju
- ▶ Kontejner će kreirati objekte, uvezati ih, konfigurisati i upravljati njihovim životnim ciklusom od kreiranja do uništenja
- ▶ Kontejner na osnovu konfiguracije koja može biti iskazana XML fajlovima, Java anotacijama ili Java kodom zna šta da radi

Spring Beans

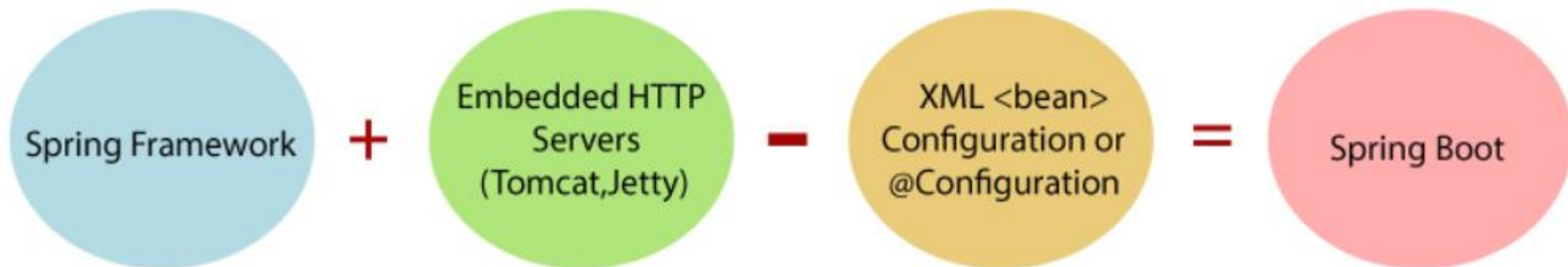
- ▶ Objekti koji formiraju aplikaciju, a koji se nalaze u nadležnosti Spring kontejnera zovu se zrna (*beans*)
- ▶ Klasa postaje bean dodavanjem konfiguracije u vidu Java anotacija ili XML metapodataka koje su potrebne Spring kontejneru da bi znao:
 - ▶ Kako da kreira bean
 - ▶ Detalje o životnom ciklusu beana
 - ▶ Zavisnosti koje bean poseduje

Spring Boot

- ▶ Open source radni okvir
- ▶ Organizovan je u oko 20 modula
- ▶ <https://spring.io/projects/spring-boot>



Spring Boot®



Konfiguracija bazirana na anotacijama

Anotacija	Opis
@Component	Stereotip za generalnu upotrebu (sve u Spring kontejneru je bean tj. komponenta)
@Controller	Označava da je klasa Spring MVC kontroler
@Repository	Označava da je klasa repozitorijum podataka
@Service	Označava da klasa sadrži neku biznis logiku

DI anotacije

- ▶ `@Autowired`, `@Inject`, `@Resource` – rade DI povezivanje i mogu se primeniti na konstruktor, set metode ili attribute
- ▶ `@Autowired` je Spring specifična i najčešće korišćena anotacija za DI
- ▶ `@Qualifier` - u kombinaciji sa `@Autowired` može se koristiti da jednoznačno naglasi koja komponenta se injektuje (ukoliko ima više komponenti koje služe istoj svrsi)