SPRING MVC FRAMEWORK

WEB PROGRAMIRANJE
NOVI SAD

IVAN PERIĆ

Spring Uvod

- Radni okvir za razvoj Java aplikacija
- Prvenstveno se koristi za razvoj serverskog dela Java veb aplikacija
- Sadrži skup biblioteka koje pružaju podršku za
  - Dependency Injection
  - Aspektno orijentisano programiranje
  - Razvoj veb aplikacija po MVC šablonu
  - Pristup bazama podataka
  - Autentikaciju i autorizaciju

SpringBoot

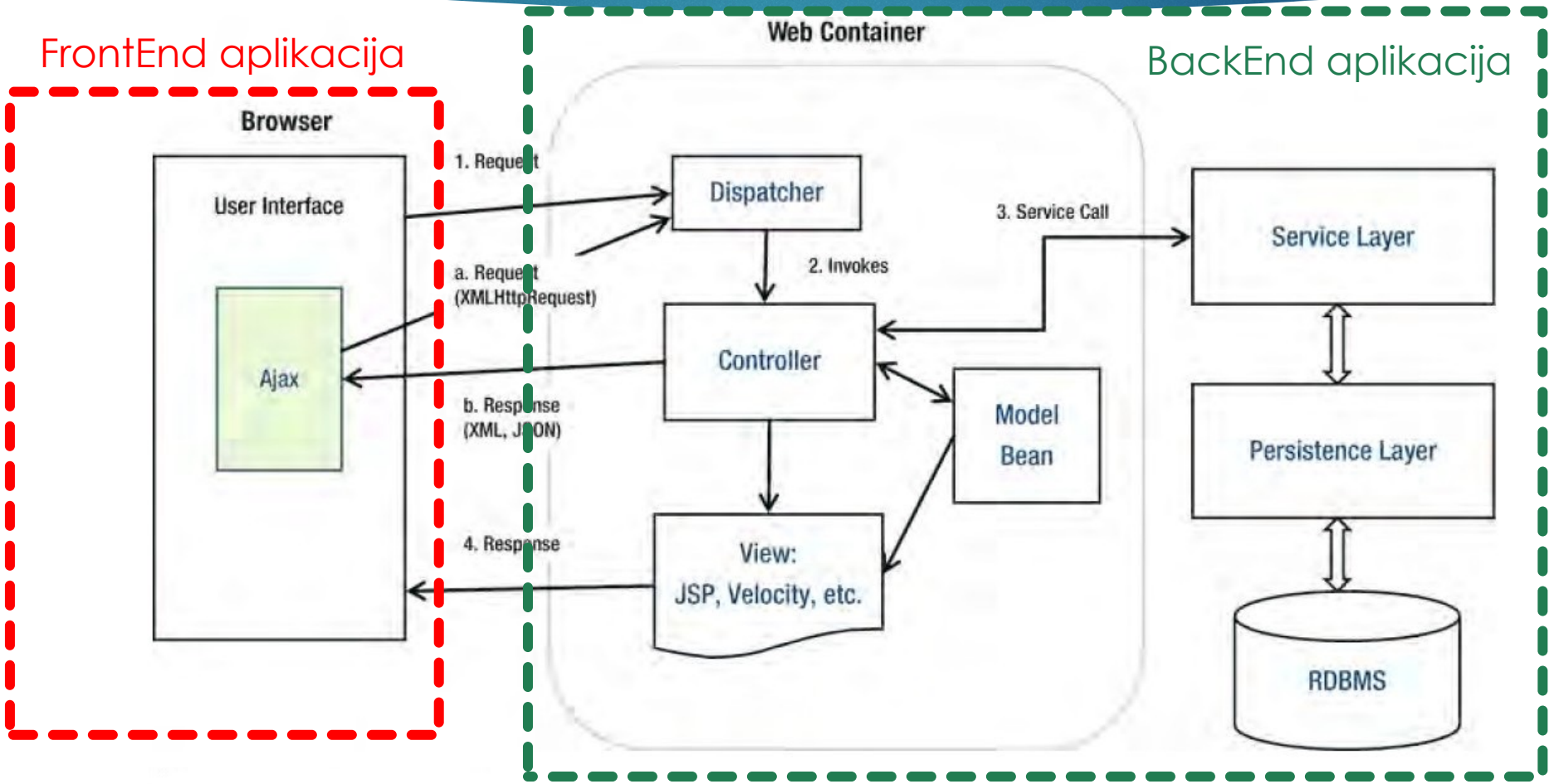
- Spring-bazirani razvojni okvir za pojednostavljeni razvoj aplikacija
- Pojednostavljuje konfigurisanje i razvoj aplikacije kroz skup gotovih rešenja
  - Jednostavnije se dobija konfigurisana Spring aplikacija
  - Jednostavnije pokretanje
    - Ugrađen veb server
  - Jednostavnije upravljanje paketima
    - Skup pripremljenih Maven artefakata
  - Konfiguriše Spring kontejner automatski gde god je moguće

- Ideja je da se programer fokusira inicijalno na razvoj aplikacije umesto na njen životni ciklus (konfiguraciju, postavljanje, upravljanje projektom, ...)

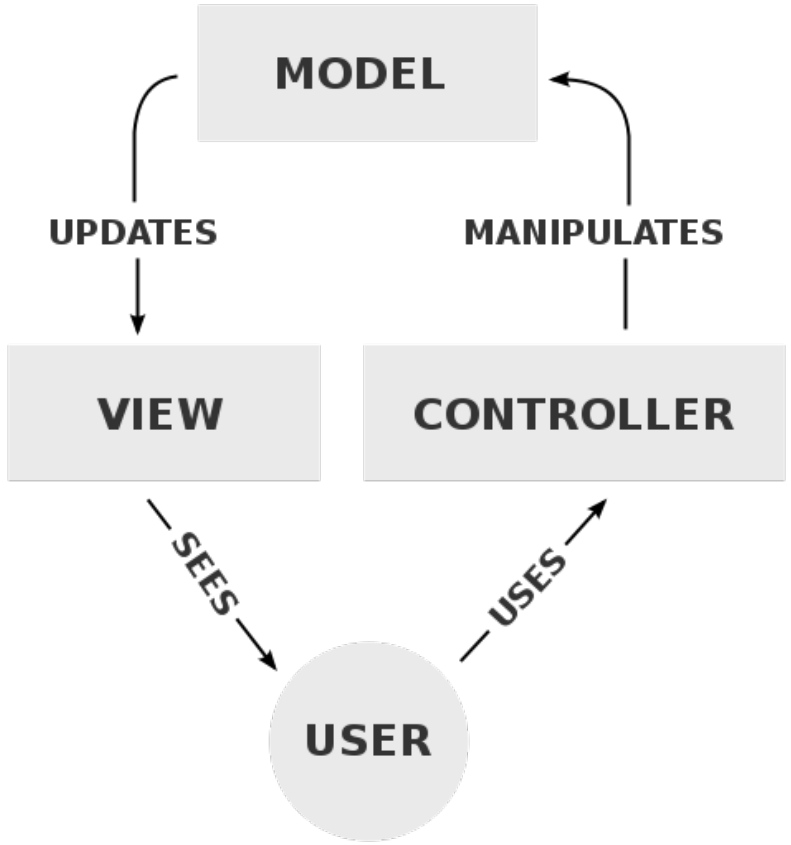
Pokretanje SpringBoot aplikacije

- Postoji klasa koja ima `main` metodu
  - Aplikacija se pokreće kao da je *stand-alone* aplikacija
- Pokreće se ugrađeni Tomcat server sa postavljenom veb aplikacijom
  - Nema potrebe pri razvoju aplikacije za odvojenim serverom i postavljanjem aplikacije u obliku war fajla
- /students-testing/src/main/java/rs/ac/uns/ftn/kts/students/StudentsApplication.java

Primer jedne arhitekture web aplikacije sa Spring backend-om

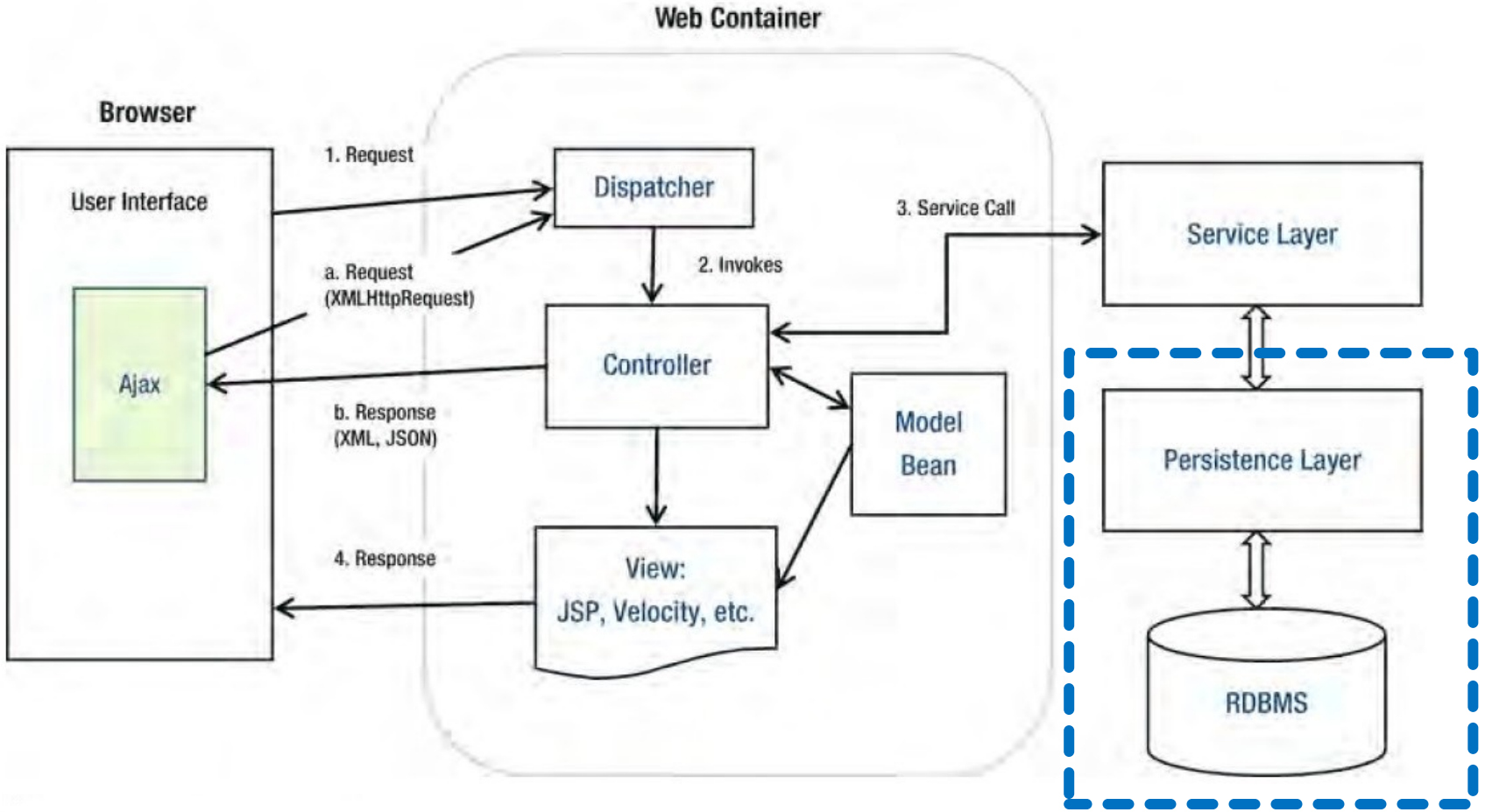


Pre početka – MVC šablon (pattern)



- MVC deli softverski sistem u tri dela:
  - **View** - obezbeđuje korisniku interfejs pomoću koga korisnik unosi podatke i/ili poziva odgovarajuće operacije koje treba da se izvrše nad model-om. View prikazuje korisniku stanje model-a.
  - **Controller** - osluškuje i prihvata zahtev od klijenta za izvršenje operacije. Nakon toga poziva operaciju koja je definisana u model-u, i ukoliko model promeni stanje, obaveštava view o promeni stanja.
  - **Model** - predstavlja stanje sistema koje mogu promeniti operacije model-a. Model ne mora da zna ko su view i controller.

Spring MVC – Sloj za perzistenciju podataka



Spring MVC – Sloj za perzistenciju podataka

- Perzistencija podataka u Spring aplikaciji se vrši putem perzistentnog sloja
- Poznatiji pod nazivom DAO (Data Access Object) sloj
- Ovaj sloj od nas „sakriva“ bazu podataka u obliku u kom je mi za sada poznajemo (skup tabela, odnosno relacija) i predstavlja je kao skup Java klasa.
- Svaki realni entitet je opisan posebnom Java klasom, a mapiranje tih klasa na našu bazu podataka se radi automatski korišćenjem mehanizama koje nudi Spring framework (ORM – objektno-relaciono mapiranje)

Spring MVC – Sloj za perzistenciju podataka

- **Entity klase (anotirane sa** @Entity**)**
  - Entity klase opisuju konkretan entitet iz realnog sveta (predstavlja klasu koja opisuje tabelu u našoj bazi podataka)
  - Entity klase služe samo da opišu konkretne entitete iz naše baze podataka
  - Primeri u paketu **„model“**: **/students-testing/src/main/java/rs/ac/uns/ftn/kts/students/model**
  - Entiry klase se u jHipster generisanoj aplikaciji nalaze u paketu **„domain“**
- **Repository interfejsi**
  - U repository interfejsima definišemo mapiranje Entity klasa na našu bazu podataka.
  - Koriste Spring Data JPA (JavaPersistenceAPI) da bi izvršile mapiranje sa objekata na relacije (ORM)
  - Dovoljno je samo da kažemo da se naša Entity klasa mapira na neku tabelu u bazi podataka tako što ćemo naslediti JpaRepository<Entity, PrimaryKeyType> interfejs.
  - Primeri u paketu **„repository“**: **/students-testing/src/main/java/rs/ac/uns/ftn/kts/students/repository**

Spring MVC – Sloj za perzistenciju podataka

- Spring Data JPA
  - Koristi JPA specifikaciju za objektno-relaciono mapiranje
  - Podrška za jednostavan razvoj sloja za pristup podacima
  - Eliminiše potrebu ponovnog pisanja sličnog koda
  - Programer samo specificira šta želi da dobije od podataka
    - samo dobavljanje će obaviti Spring Data JPA

Spring MVC – Query metode nad repozitorijumom

- Ideja je da se poštovanjem konvencije u imenovanju metode, metoda samo deklariše
  - Na osnovu deklaracije koja poštuje specificiranu formu, Spring automatski obezbeđuje implementaciju

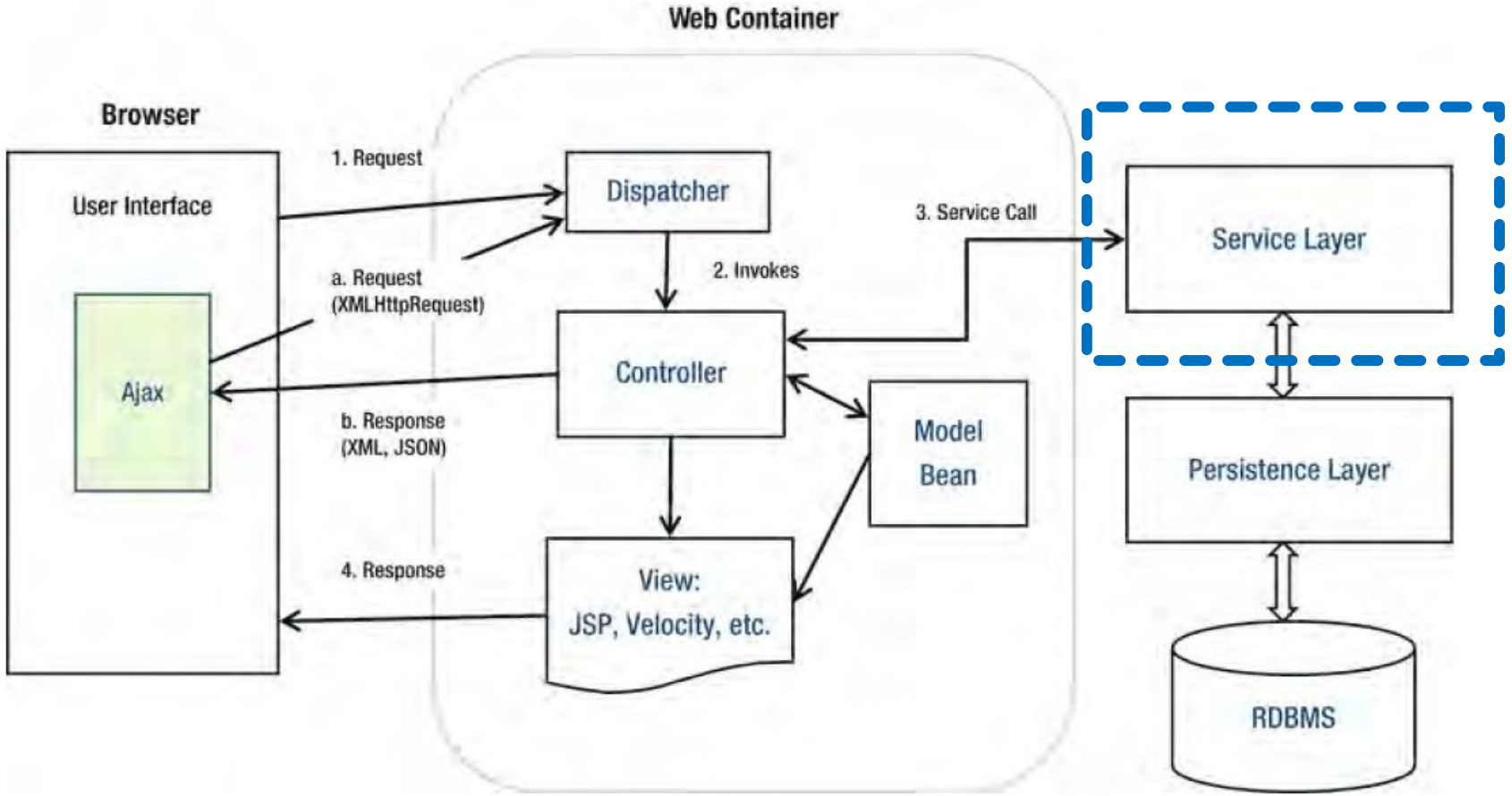
```
public interface PersonRepository extends Repository<User, Long> {

  List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
  List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);
  // Enabling ignoring case for all suitable properties
  List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
  List<Person> findByLastnameOrderByFirstnameDesc(String lastname);
}
```

Spring MVC – Query metode nad repozitorijumom

- Moguće je i specijalnim parametrima zahtevati sortiranje i paginaciju i dobiti jednu stranicu podataka

```
Page<User> findByLastname(String lastname, Pageable pageable);
```

Spring MVC – Servisni sloj



Spring MVC – Servisni sloj

- Servisni sloj se koristi kao omotač (wrapper) DAO sloja i predstavlja opšteprihvaćen način za korišćenje DAO sloja
- Može biti izostavljen tako što ćemo koristiti DAO sloj direktno, ali njegovo uvođenje može doneti mnoge pogodnosti:
  - U servisni sloj se stavljaju sve CRUD (create, retrieve, update, delete) metode koje repositoy nudi. Zašto ne koristiti direktno repository? Repository nudi samo osnovne operacije. Ako želimo da proverimo polovnu logiku u tim podacima to nije moguće. Takve provere možemo vršiti u metodama u servisnom sloju
- Preporuka je da se u Service sloju piše poslovna logika sistema

- Primeri u paketu **„service“** : /students-testing/src/main/java/rs/ac/uns/ftn/kts/students/service

Spring MVC – Servisni sloj

```
@Service
public class StudentService {

    @Autowired
    StudentRepository studentRepository;

    public Student findOne(Long id) {
    return studentRepository.findOne(id);
    }

    public List<Student> findAll() {
    return studentRepository.findAll();
    }
}
```

Anotacija za servise

Ovaj objekat nikada nije instanciran sa
StudentRepository studentRepository = new StudentRepository();

Kako možemo koristiti objekat za koji nikada nije zauzeta memorija i jednak je „null“? Zar nećemo izazvati Exception?

@Autowired anotacija rešava problem i ovaj mehanizam je poznat pod nazivom **DependencyInjection**.

ćeni

nata

n

Primer metode koja iz repozitorijuma svih studenata preuzima sve studente

Dependency injection

- Poznato i pod nazivom Inverzija kontrole
- Spring kontejner upravlja životnim ciklusom objekata
- Programer piše programski kod u kojem samo koristi objekte
- Spring kontejner je zadužen za
  - kreiranje,
  - inicijalizaciju,
  - konfigurisanje i
  - obezbeđivanje objekata dostupnim
- **Objekti kojima kontejner upravlja se nazivaju *Beans***
- *U našem slučaju se kao Spring kontejner koristi Tomcat i on će sam automatski inicijalizovati sve objekte anotirane sa @Autowired*

Definicija Bean klasa

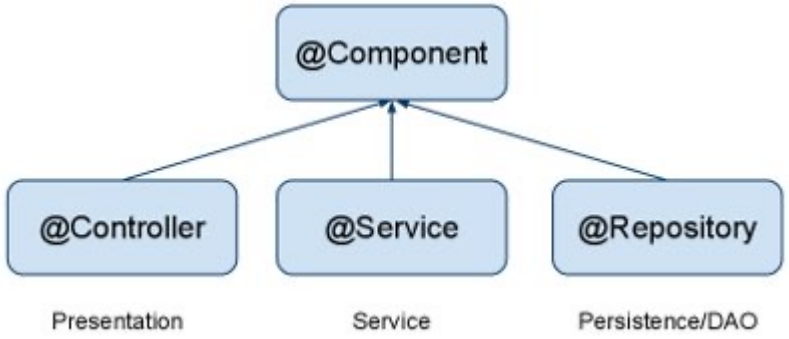
- JavaBean je običan standard za pisanje klasa.

- Standard nalaže da klasa mora zadovoljavati sledeće osobine da bi bila **Bean**:
  - **Svi atributi** klase moraju biti privatni (**private**) – koristiti getere i setere
  - Klasa mora imati javni (**public**) **konstruktor bez parametara**
  - Klasa implementira interfejs **Serializable**

- Mnogi razvojni okviri zahtevaju da klase budu pisane uz poštovanje ovog standarda.

Definicija Bean klasa

- Gde se označava da je neka klasa *Bean* i da će objektima te klase biti upravljano od strane kontejnera?
  - Anotacijom nad klasom se klasa proglašava Spring Bean klasom

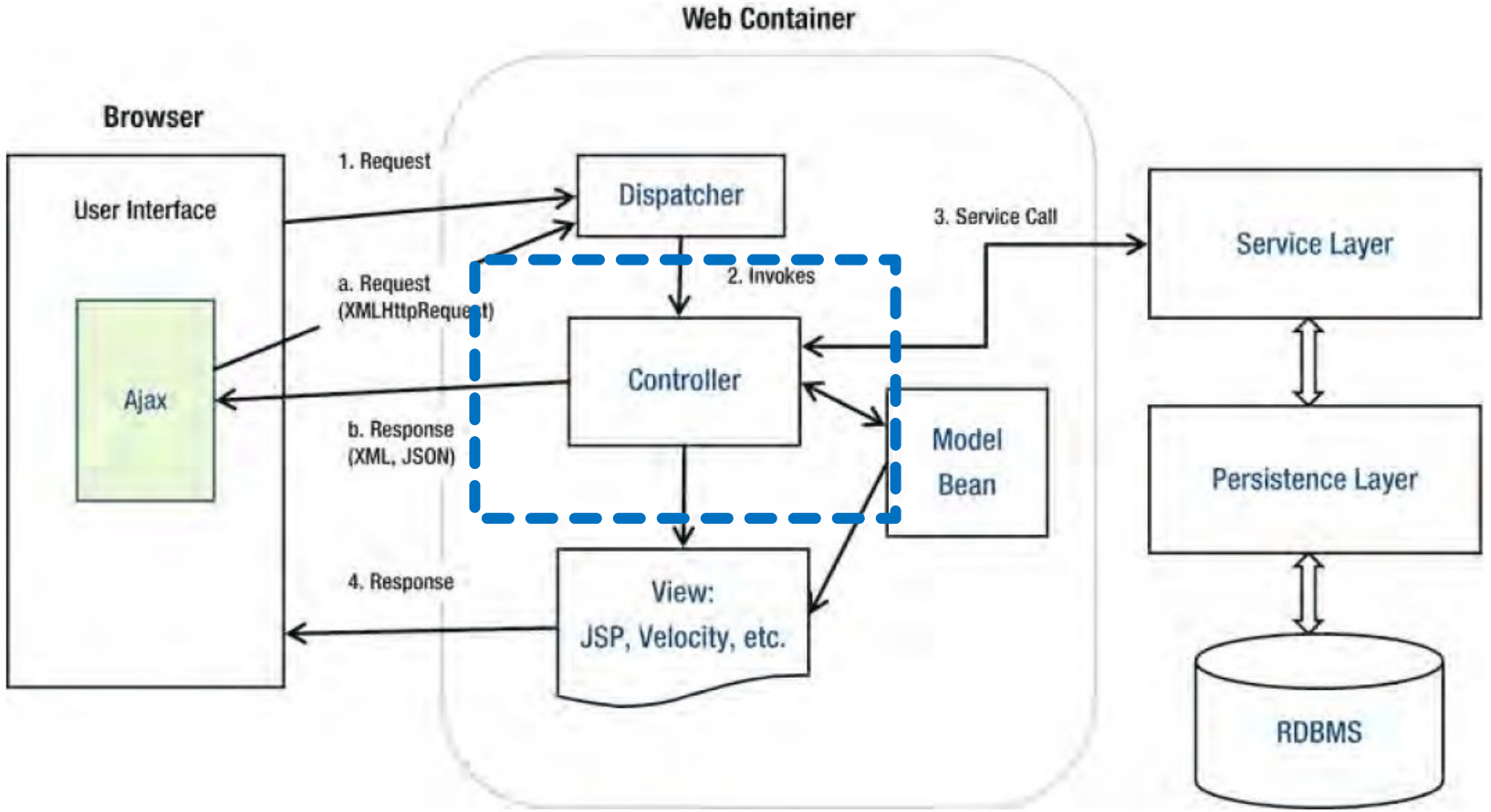


- Ako želimo da klasa bude automatski pronađena i prepoznata od strane kontejnera kao *bean*
  - Osnovna anotacija `@Component`

Umesto generičke anotacije `@Component` u praksi se koriste njene specijalizacije zavisno od uloge klase u aplikaciji

- Za automatsko pronalaženje klasa potrebno je na nivou aplikacije postaviti anotaciju `@ComponentScan`

Spring MVC – Sloj za komunikaciju



Sloj za komunikaciju

- Implementira se kao skup web servisa
- Web servisi se danas najčešće realizuju korišćenjem REST softverske arhitekture
  - RESTful web servisi
- Klase anotirane sa @Controller ili @RestController

Sloj za komunikaciju

- Servis se definiše kreiranjem klase anotirane kao **`@RestController`**
- Servis će biti **javno dostupan putem URL definisanog anotacijom `@RequestMapping`**
- Metode servisa se implementiraju kao metode klase
  - `@RequestMapping` anotacija definiše putanju do konkretne metode
  - Putanja može da sadrži dinamičke vrednosti
    - Metodi se automatski prosleđuju kao parametri označeni anotacijom `@PathVariable`

Sloj za komunikaciju

- Povratna vrednost metode se serijalizuje u željeni format za prenos preko mreže
  - Povratna vrednost su objekti klase `ResponseEntity`
- Parametri HTTP zahteva se automatski parsiraju
  - Kriera se parametar metode
  - Parametar se označeni anotacijom `@RequestParam`
- Telo HTTP zahteva se automatski parsira
  - Kreira se parametar metode
  - Parametar se označi anotacijom `@RequestBody`
- Parametri i telo se automatski deserijalizuju u tip koji odgovara tipu parametra metode
- Primeri u paketu **„controller“**: /students-testing/src/main/java/rs/ac/uns/ftn/kts/students/web/controller
- Kontroler klase se u jHipster generisanoj aplikaciji nalaze u paketu **„web.rest“**

Sloj za komunikaciju

```java
@RestController
@RequestMapping(value="api/courses")
public class CourseController {
    @Autowired
    private CourseService courseService;

    @RequestMapping(method = RequestMethod.GET)
    public ResponseEntity<List<CourseDTO>> getCourses() {
        List<Course> courses = courseService.findAll();
        //convert courses to DTOs
        List<CourseDTO> coursesDTO = new ArrayList<>();
        for (Course s : courses) {
            coursesDTO.add(new CourseDTO(s));
        }
        return new ResponseEntity<>(coursesDTO, HttpStatus.OK);
    }

    @RequestMapping(value="/{id}", method=RequestMethod.GET)
    public ResponseEntity<CourseDTO> getCourse(@PathVariable Long id){
        Course course = courseService.findOne(id);
        if(course == null){
            return new ResponseEntity<>(HttpStatus.NOT_FOUND);
        }

        return new ResponseEntity<>(new CourseDTO(course), HttpStatus.OK);
    }
}
```

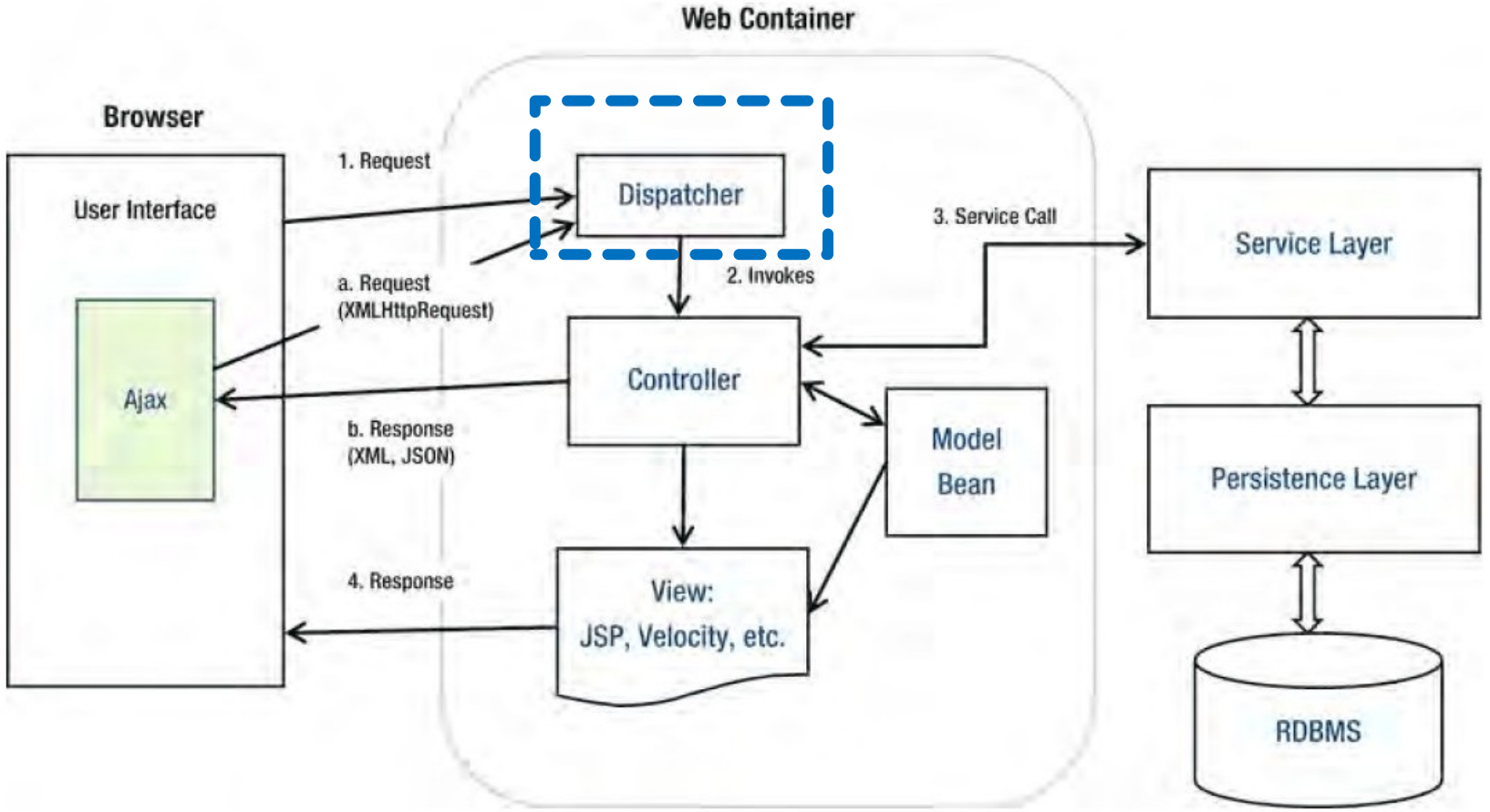
Adresa svake metode koja se nalazi unutar ovog kontrolera počinje sa *localhost:8080/api/courses*

Ova metoda nema @RequestMapping anotaciju što znači da će njena adresa biti ista kao adresa kontrolera u okviru koga se nalazi. Takođe, ova metoda će biti pozvana samo ako je tip HTTP zahteva GET

Ova metoda ima @RequestMapping anotaciju što znači da će njena adresa biti ista kao **adresa kontrolera + proširenje adrese navedeno u anotaciji same metode** (localhost:8080/api/courses/{id})

Npr. ukoliko korisnik pošalje get zahtev na adresu: localhost:8080/api/courses/5, dobiće kurs sa id=5

Spring MVC


Web Container
Browser
User Interface
Ajax
1. Request
a. Request
(XMLHttpRequest)
b. Response
(XML, JSON)
4. Response
Dispatcher
2. Invokes
Controller
Model
Bean
View:
JSP, Velocity, etc.
3. Service Call
Service Layer
Persistence Layer
RDBMS

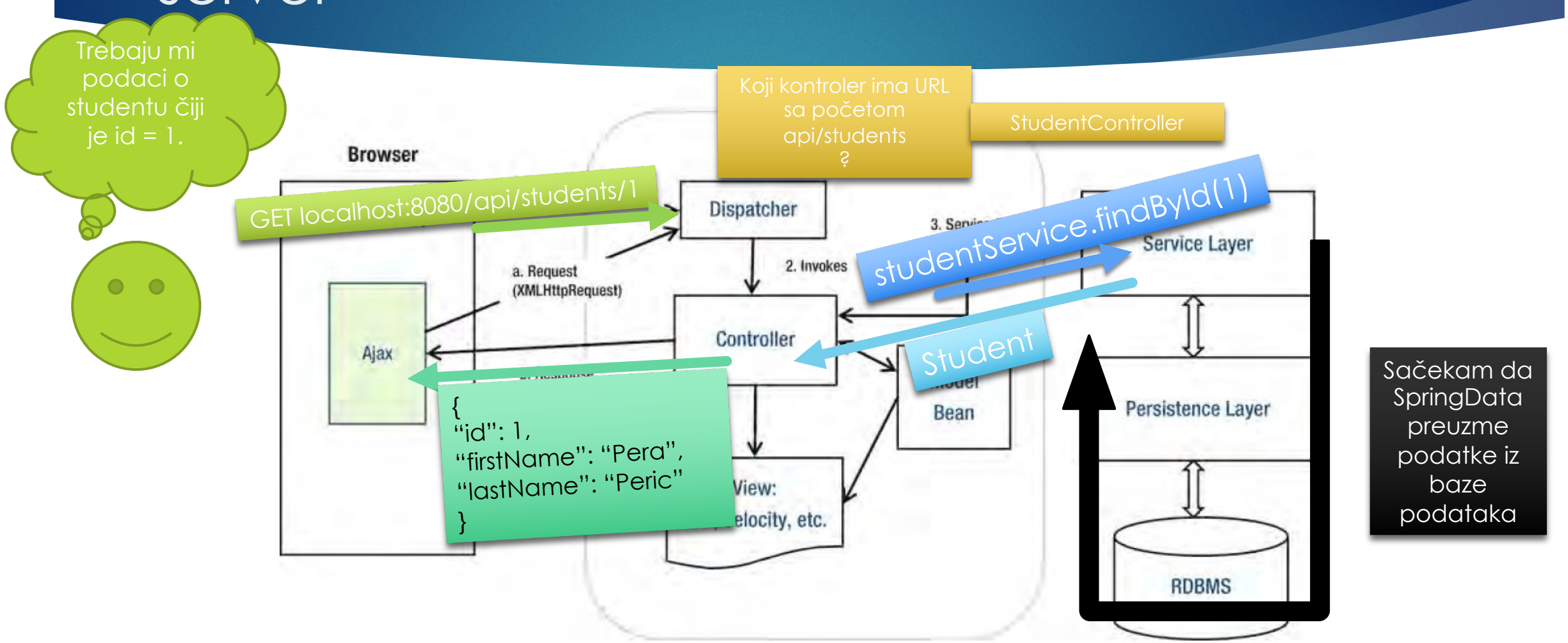
Spring MVC - Dispatcher

- Svi HTTP zahtevi koji stižu ka Spring aplikaciji u suštini gađaju **Dispetcher servlet**.
- Kada Dispetcher dobije zahtev, on na osnovu njegovo URL-a odredi koji kontroler treba da bude pozvan i njemu prosleđuje zahtev.
- U kontroleru se onda donosi odluka koja metoda unutar njega će biti pozvana na osnovu tipa zahteva (GET, POST, PUT,...) i na osnovu njenog URL-a

- Odgovor iz kontrolera može biti čist JSON ili XML, a kontroler može da vraća i kompletnu web stranicu koja treba da se servira klijentu. Pošto kreiramo REST komunikaciju to neće biti slučaj i naša Spring aplikacija će vraćati JSON stringove.

Primer jednog ciklusa komunikacije klijent-server

Trebaju mi podaci o studentu čiji je id = 1.



Sačekam da SpringData preuzme podatke iz baze podataka