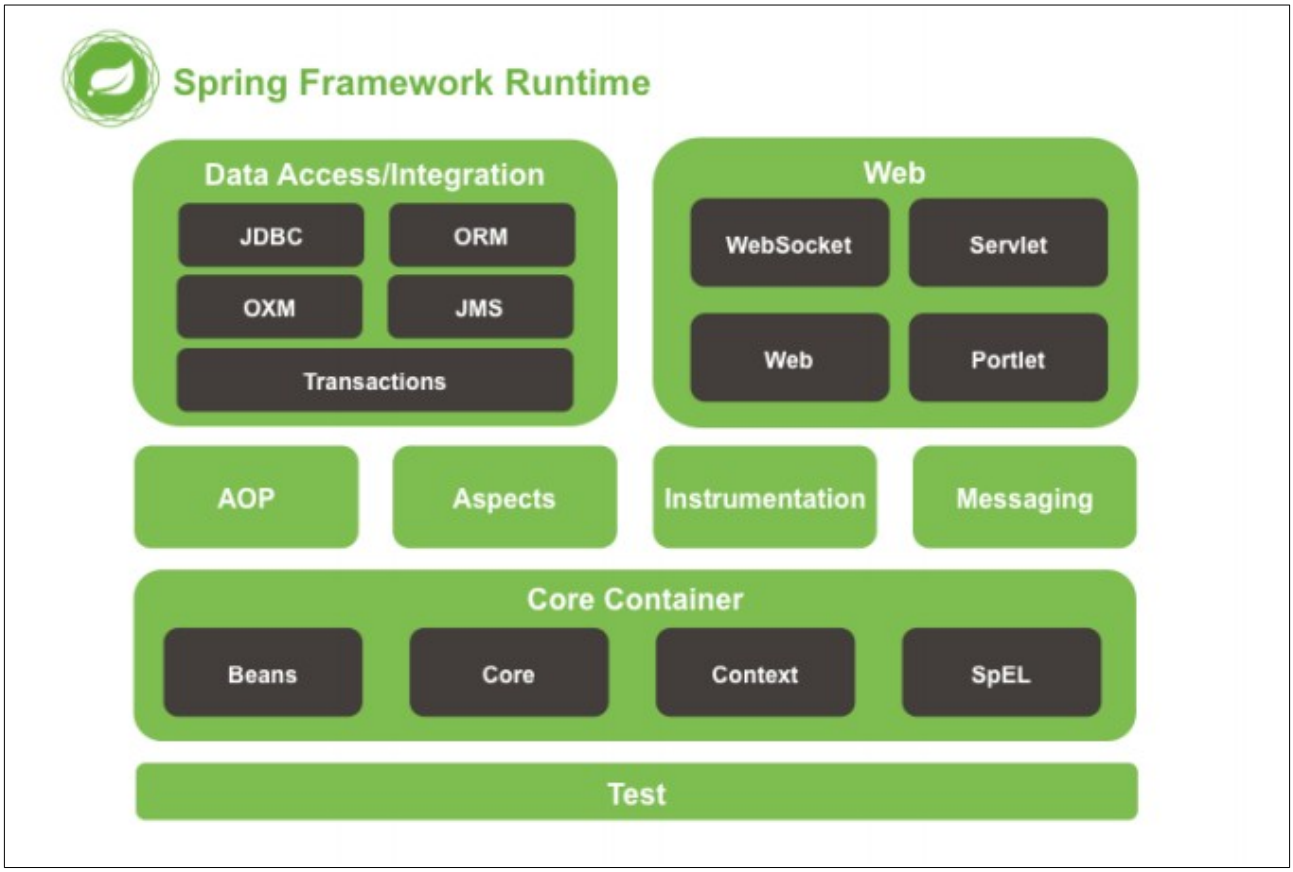
Spring Framework
Spring Boot
Spring MVC

SADRZAJ

- *lightweight* Java bazirano razvojno okruženje
- za razliku od mnogih drugih Java razvojnih okruženja, može se koristiti za razvoj bilo kakvih Java baziranih aplikacija
  - desktop, web ili Java EE aplikacije
- nudi veoma bogat skup funkkionalnosti koje su grupisane po modulima
- moguće je iskoristiti samo one module koji su potrebni za razvoj aplikacije
- prvu verziju Spring-a razvio je Rod Johnson 2002. godine
- Spring se konstantno razvija
- veliki broj projekata razvijenih korištenjem Spring framework-a
- velika zajednica Spring programera
- najpopularnije Java razvojno okruženje

- 0.9 – 2002
- 1.0 – 2003
- 2.0 – 2006
- 3.0 – 2009
- 4.0 – 2013
- 5.0 – 2017
- trenutna verzija – 5.3.1, novembar 2020. godine

Spring Services



Spring razvojno okruženje se sastoji od oko 20 modula svaki od ovih modula se može koristiti u bilo kojim Java aplikacijama što omogućava korištenje samo onih modula koji su potrebni za razvoj aplikacije, dok ostale module nije potrebno učitavati

Core Container se sastoji od 4 modula: Core, Beans, Context i SpEL

- Core i Beans modul predstavljaju fundamentalni dio razvojnog okruženja koji uključuju IoC (*Inversion of Control*) i DI (*Dependency Injection*)
- Modul Context služi za pristupanje objektima instanciranim pomoću IoC kontrolera. Osim toga pruža mogućnosti internacionalizacije (npr. učitavajući datoteke u kojima se nalazi lokalizovane vrijednosti), propagaciju događaja, učitavanje resursa, itd.
- SpEL je jezik za manipulaciju objektima u toku izvršavanja aplikacije. Omogućava pisanje i čitanje atributa objekata, pozivanje i izvršavanje metoda, pristupanje elementima kolekcija, sadrži logičke i aritmetičke izraze itd.

IoC predstavlja princip programiranja pomoću kojeg se upravljanje objektima (instanciranje objekata, rukovanje životnim ciklusom, upravljanje međusobnim zavisnostima itd.) vrši u pozadini od strane programskog jezika ili razvojnog okruženja

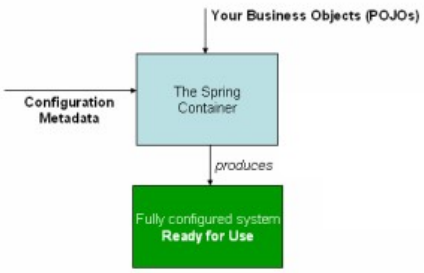
da bi programski jezik ili razvojno okruženje bilo u mogućnosti vršiti upravljanje objektima, potrebno je izvršiti određena podešavanja na način koji sam programski jezik ili razvojno okruženje nalaže

- najčešće je potrebno specificirati koji objekat će biti upravljan u pozadini od strane programskog jezika (razvojnog okruženja), kakav treba da bude njegov životni ciklus, koje su njegove zavisnosti i u kom trenutku je potrebno tu zavisnost ostvariti (ubrizgati)

Spring IoC kontejner dobija instrukcije o objektima koje je potrebno instancirati, njihovim podešavanjima i definicijama zavisnosti tako što vrši čitanje konfiguracionih metapodataka zapisanih od strane programera

konfiguracioni metapodaci se mogu specifikovati na tri načina pomoću:

- XML konfiguracionih datoteka (postoji od prve verzije Spring-a),
- Java anotacija (uvedene su u Spring 2.5 verziji) i
- Java programskog koda (od Spring 3.0 verzije)



U standardnom aplikacijama, nije potrebno vršiti dodatna podešavanja da bi se instancirao objekat Spring IoC kontejnera, dok je u web aplikacijama potrebno definisati kontejner u web.xml datoteci

- Spring Boot je ekstenzija Spring razvojnog okruženja
- omogućava razvoj aplikacije sa minimalnom konfiguracijom (ili bez nje)
  - autokonfiguracija – posebna karakteristika Spring Boot-a
- skraćuje broj linija koda
- olakšava razvoj web aplikacija
- ne zahtijeva XML konfiguraciju
- ugrađeni aplikativni server (Tomcat, Jetty ili Undertow) – nema potrebe za WAR arhivama
- obezbjeđuje određene *production-ready* mogućnosti
- skraćuje vrijeme potrebno za razvoj aplikacije
- jednostavnije pokretanje
- najčešće se koristi za razvoj REST API-ja
- trenutna verzija 2.4.2

Spring Initializr je alat za generisanje strukture Spring Boot projekta

https://start.spring.io/

https://start-scs.cfapps.io/

podrška u razvojnim alatima: STS (*Spring Tool Suite*), IntelliJ IDEA, NetBeans, Eclipse

kreiranje projekta

- Projekat: definiše vrstu projekta (Maven Project ili Gradle Project)
- Jezik: izbor programskog jezika: Java, Kotlin i Groovy
- Spring Boot: Spring Boot verzija
- Metapodaci o projektu: informacije povezane sa projektom (Group, Artifact, Name, Description)
- Zavisnosti: kolekcija artefakata koje dodajemo u projekat

Spring anotacije

- @SpringBootApplication – omogućava automatsku konfiguraciju Spring Boot okruženja i skeniranje komponenata
- @Component označava komponentu upravljanu od strane Spring okruženja
- pretvara klasu u Spring bean u vrijeme autoskeniranja
- klase dekorisane ovom anotacijom su kandidati za automatsku detekciju
- @Repository, @Service i @Controller su specijalizacije @Component
- @Service – označava da je klasa servisna klasa
- @Repository – označava da je označena klasa repozitorijum, tj. da klasa koja apstrahuje pristup skladištu podataka
- @Controller – označava web kontroler, tj. klasu koja je sposobna da obrađuje zahtjeve

@Bean – označava da metoda kreira bean kojim će upravljati Spring (Spring bean)

@Configuration – označava da se radi o konfiguracionoj klasi koja može sadržati definicije bean-ova

@Autowired – označava da konstruktor, polje ili metoda bude automatski povezana DI mehanizmom Spring okruženja

@Qualifier – označava koji bean će biti injektovan u slučaju autowire-inga gdje postoji više od jednog bean-a istog tipa

@Value – označava podrazumijevanu vrijednost polja ili parametra metode / konstruktora

@Lazy – označava tzv. „lijenu“ inicijalizaciju bean-a

@DependsOn – označava bean od kojeg zavisi tekući bean. Garantuje se da će specificirani bean biti kreiran prije tekućeg bean-a.

- @RequestMapping – mapira HTTP zahtjev u metodu kontrolera
- @GetMapping – mapira HTTP GET zahtjev u metodu kontrolera
  - isto kao @RequestMapping (method = RequestMethod.GET)
- @PostMapping – mapira HTTP POST zahtjev u metodu kontrolera
  - isto kao @RequestMapping (method = RequestMethod.POST)
- @PutMapping – mapira HTTP PUT zahtjev u metodu kontrolera
  - isto kao @RequestMapping (method = RequestMethod.PUT)
- @DeleteMapping – mapira HTTP DELETE zahtjev u metodu kontrolera
  - isto kao @RequestMapping (method = RequestMethod.DELETE)
- @PatchMapping – mapira HTTP PATCH zahtjev u metodu kontrolera
  - isto kao @RequestMapping (method = RequestMethod.PATCH)
- @CookieValue – označava da parametar metode treba da sadrži vrijednost specificiranog cookie-ja
- @CrossOrigin – označava da su mogući zahtjevi između različitih domena (JS se servira sa sa jednog servera, a podaci sa drugog). Koristi se i na nivou klase i na nivou metode. Može se specificirati domen sa kojeg je moguće uputiti zahtjeve (podrazumijevano su to svi domeni).

- @PathVariable
  - označava argumente metode čije vrijednosti se preuzimaju iz dijela URL-a
- @RequestParam
  - označava argumente metode čije vrijednosti se preuzimaju iz *query* stringa
- @RequestAttribute
  - označava argumente metode čije vrijednosti se preuzimaju iz atributa (koji su kreirani u istom HTTP zahtjevu, npr. u nekom filteru)

@RequestBody

- označava argumente metode čije vrijednosti se preuzimaju iz tijela HTTP zahtjeva

@RequestHeader

- označava argumente metode čije vrijednosti se preuzimaju iz zaglavlja HTTP zahtjeva

@ResponseBody

- označava da rezultat izvršavanja metode kontrolera treba da bude vraćen u tijelu odgovora u specificiranom formatu (npr. JSON ili XML)

@ResponseStatus

- označava da metoda ili klasa izuzetka vraća odgovor sa specificiranim statusnim kodom i opisom

@ExceptionHandler

- koristi se na nivou metode da označi da metoda obrađuje izuzetak koji može biti bačen unutar kontrolera
- kao argument se navodi klasa izuzetka koji će se hvatati ili niz klasa izuzetaka koji će se hvatati

@ControllerAdvice

- označava klasu koja posjeduje metode anotirane sa @ExceptionHandler, @InitBinder i @ModelAttribute koje se odnose na sve @RequestMapping metode
- na ovaj način je moguće imati jednu klasu sa većim brojem metoda, gdje svaka metoda obrađuje isti izuzetak koji može biti bačen unutar različitih metoda kontrolera

@RestController

- označava klasu čije metode vraćaju objekte umjesto pogleda (view-a). @RequestMapping metode klase koja je anotirana ovom anotacijom nije potrebno anotirati sa @ResponseBody
- @RestController kombinuje @Controller i @ResponseBody

@SessionAttribute

- označava da se parametar metode veže za atribut sesije, tj. čini da je parametar metode dostupan unutar sesije

@SessionAttributes

- označava da JavaBean objekat treba biti dodan u sesiju. Koristi se zajedno sa @ModelAttribute anotacijom.

- **@Query**
  - označava SQL upit koji treba da bude izvršen
- **@Transactional**
  - označava transakciju u bazi podataka
- **@Modifying**
  - označava da SQL upit modifikuje podatke (putem INSERT, UPDATE, DELETE ili neke DDL naredbe)
- **@Param**
  - označava imenovani parametar metode čiju vrijednost možemo proslijediti u SQL upit
- **@Procedure**
  - označava pozivanje uskladištene procedure
- **@Lock**
  - označava mod zaključavanja tabele

@Id
- označava da je polje modela (entiteta) primarni ključ

@Transient
- označava da je polje modela (entiteta) tranzijentno

@CreatedBy
- označava principal-a koji je kreirao objekat

@LastModifiedBy
- označava principal-a koji je posljednji modifikovao objekat

@CreatedDate
- označava vrijeme kreiranje objekta

@LastModifiedDate
- označava vrijeme posljednje modifikacije objekta

@PreFilter
- označava pravila za filtriranje liste koja se prosljeđuje kao parametar metode

@PostFilter
- označava pravila za filtriranje rezultata metode (liste koju metoda vraća)

@Secured

- označava listu rola koje mogu izvršiti metodu

@RoleAllowed

- kao i @Secured

@PreAuthorize

- označava uslov koji mora biti ispunjen da bi korisnik bio autorizovan za izvršavanje metode, tj. upita

@PostAuthorize

- označava uslov koji mora biti ispunjen da bi korisnik bio autorizovan, ali nakon izvršene metode, tj. upita. Ovo se koristi u slučaju kada metoda vraća rezultat koji se koristi za evaluaciju uslova.

Spring APPLICATION PROPERTIES

Spring Boot Framevork posjeduje ugrađeni mehanizam za konfiguraciju aplikacije

- application.properties
- ova datoteka se nalazi unutar direktorijuma src/main/resources
  - #ime aplikacije
  - spring.application.name = Ime aplikacije
  - #serverski port
  - server.port = 8081

mehanizam koji omogućava dodavanje biblioteka u classpath

omogućava brži i jednostavniji razvoj

imenovanje starter-a:

- spring-boot-starter-*

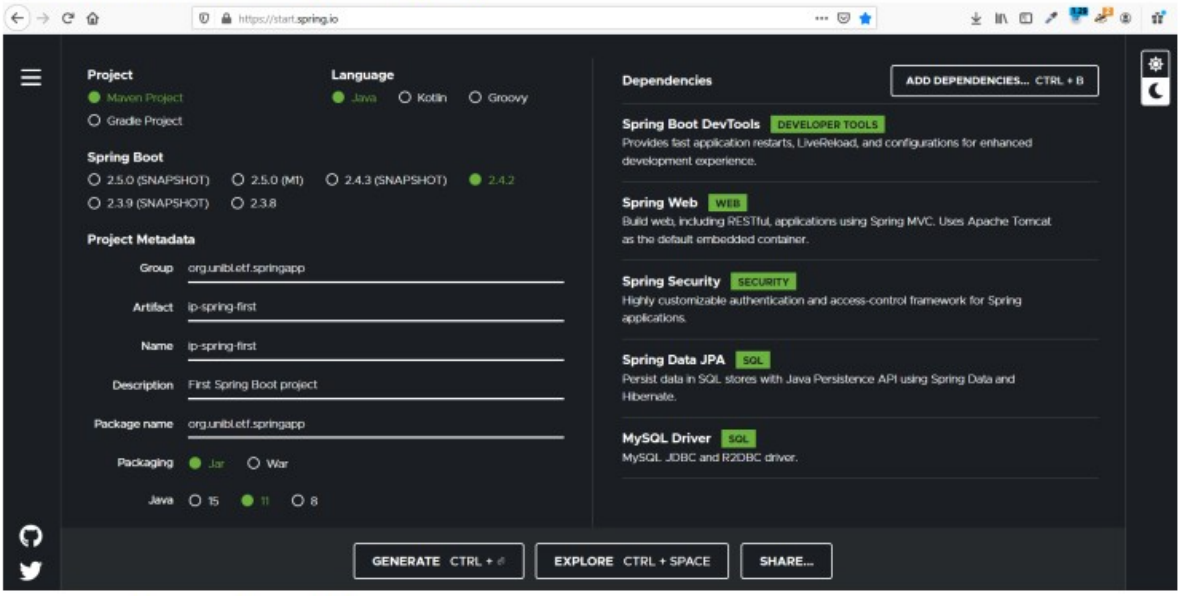
primjeri:

- spring-boot-starter-web-services
- spring-boot-starter-data-redis
- spring-boot-starter-web
- spring-boot-starter-activemq
- spring-boot-starter-data-elasticsearch
- spring-boot-starter-jdbc
- spring-boot-starter-jersey
- spring-boot-starter-data-neo4j
- spring-boot-starter-data-ldap
- spring-boot-starter-security
- spring-boot-starter-data-jpa
- ...

Sprring Initializer

Spring Initializr

- https://start.spring.io/



- raspakovati arhivu
- importovati Maven projekat
- Maven -> "Update project"

REST Controller

```java
@RestController
@RequestMapping("/api/v1/books")
public class BookController {

    @Autowired
    private BookService bookService;

    @GetMapping
    public ResponseEntity<List<Book>> getAllBooks(){
        List<Book> books = bookService.getAllBooks();
        return ResponseEntity.ok(books);
    }

    @GetMapping("/{id}")
    public ResponseEntity<Book> getBookById(@PathVariable Long id) throws ResourceNotFoundException{
        Book book = bookService.getBookById(id);
        return ResponseEntity.ok(book);
    }

    @PostMapping
    public ResponseEntity<?> addBook(@RequestBody Book book) {
        Book newBook = bookService.addBook(book);
        URI location = ServletUriComponentsBuilder
                .fromCurrentRequest().path("/{id}")
                .buildAndExpand(newBook.getId()).toUri();
        return ResponseEntity.created(location).build();
    }
```

Servis

```java
@Service
public class BookService {

    @Autowired
    private BookRepository bookRepository;

    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }

    public Book getBookById(Long id) throws ResourceNotFoundException {
        Book book = bookRepository.findById(id)
                .orElseThrow(() -> new ResourceNotFoundException("Book not found - id: " + id));
        return book;
    }

    public Book addBook(@Valid Book book) {
        return bookRepository.save(book);
    }

    public Book updateBook(@Valid Book book) {
        return bookRepository.save(book);
    }

    public List<Book> getNewerThan(Integer year) {
        return bookRepository.getNewerThan(year);
    }
}
```

Repozitorijum

```
public interface BookRepository extends JpaRepository<Book, Long> {

    @Query("select b from Book b where b.year > :year")
    List<Book> getNewerThan(@Param("year") Integer year);
}
```

Entitet

```
@Entity
public class Book {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @NotBlank(message = "Author can't be blank!")
    @Size(min = 3, max = 45)
    private String author;
    @NotBlank(message = "Title can't be blank!")
    @Size(min = 3, max = 45)
    private String title;
    @NotBlank(message = "Publisher can't be blank!")
    @Size(min = 3, max = 45)
    private String publisher;
    @Min(value = 1900, message = "Min value for Year is 1900!")
    @Max(value = 2021, message = "Max value for Year is 2021!")
    private int year;
    @NotBlank(message = "ISBN can't be blank!")
    @Size(min = 3, max = 45)
    private String isbn;


    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
```

Exception Handler

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ResponseBody
    @ExceptionHandler(ResourceNotFoundException.class)
    @ResponseStatus(HttpStatus.NOT_FOUND)
    public ResponseEntity<MessageResponse> resourceNotFoundException(ResourceNotFoundException ex) {
        return ResponseEntity.status(HttpStatus.NOT_FOUND).body(new MessageResponse(ex.getMessage()));
    }

    @ResponseBody
    @ExceptionHandler(Exception.class)
    @ResponseStatus(HttpStatus.INTERNAL_SERVER_ERROR)
    public ResponseEntity<MessageResponse> globalExcpetionHandler(Exception ex) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR).body(new MessageResponse(ex.getMessage()));
    }

}
```

Securit

```
@EnableWebSecurity
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().disable();
        http.authorizeRequests().anyRequest().permitAll();
    }

}
```

```
protected void configure(HttpSecurity http) throws Exception {
  http.antMatcher("/match1/**")
    .authorizeRequests()
      .antMatchers("/match1/user").hasRole("USER")
      .antMatchers("/match1/spam").hasRole("SPAM")
      .anyRequest().authenticated();
}
```

```
@Service
public class MyService {

  @Secured("ROLE_USER")
  public String secure() {
    return "Hello Security";
  }
```

autentikacija
- password enkoderi (bcrypt, scrypt, sha256,...)

Cross Site Request Forgery (CSRF) zaštita

Security HTTP Response Headers
- Cache-Control, HSTS, X-XSS-Protection, ...

HTTP podešavanja
- redirekcija na HTTPS, HTTP STS, konfiguracija proxy servera

Form, HTTP Basic, HTTP Digest, In-Memory autentikacija, JDBC autentikacija, OAuth 2.0, LDAP, OpenID, CAS, X.509 autentikacija, SAML2...

Security filteri

JWT - JSON Web Token
- RFC 7519, 7797, 8725