SPRING FRAMEWORK

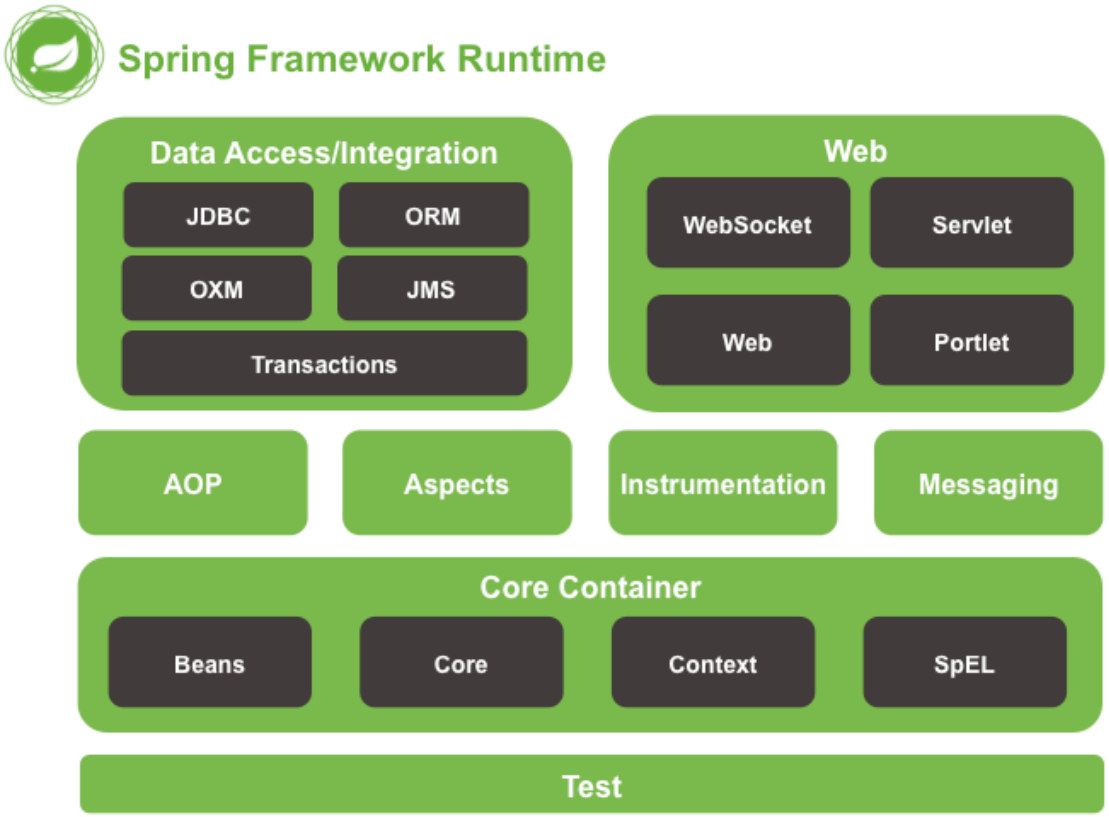
MI A SPRING?

SPRING FRAMEWORK

- Egyszerűbbé teszi a webalkalmazások készítését Java nyelven
- A Java EE (javax.* package) által nyújtott megoldásokat hivatott leegyszerűsíteni
- Moduláris - már bőven Java 10 előtt is az volt
- Régebben Servlet Container volt(.war, XML), manapság inkább monolit webservice-ek és microservice-ek írására használják(.jar, REST)
- Servlet - objektum ami HTTP requesteket fogad, HTTP response-okat szolgáltat, jellemzően a response egy XML


NEM

SPRING FRAMEWORK



spring

spring boot

SPRING FRAMEWORK

- Régebbi megközelítésben a servlet egy JSP renderelt formáját adta vissza response-ként
- Manapság, főként REST endpointokat írunk, amelyek jellemzően JSON formátumú adatot szolgáltatnak - persze van kivétel is, de az sem HTML!
- Konfigurációja végezhető XML-ben vagy Java-ban, annotációk segítségével, ezek akár vegyíthetőek is
- Legnagyobb előnye a kiadásakor a Dependency Injection volt
- MVC tervezési mintát használ, viszont ezen felül bármilyen mintával szabadon bővíthető

SPRING BOOT

SPRING BOOT

- A Spring Boot lényegében egy auto konfigurációt szolgáltató lehetőség a Spring Framework-höz
- Így elhagyható az XML alapú konfiguráció 🤩 és alkalmazható .properties, .yml alapú, ahol lényegében előre kínált, beállítási lehetőségeken tudunk módosítani, ezen felül természetesen elérhető maradt az annotáció alapú config is
- Ami miatt még fontos: átalakult a modularizációs rendszer, úgy, hogy a lehető legkisebb esély legyen a verzió különbségekből adódó hibákra
    - Már nem egyszerű servlet containereket csinálunk, hanem teljesértékű webserviceket - pl. Embeded Tomcat segítségével
- Így gyorsan és egyszerűen tudunk webappot készíteni: start.spring.io

HOGY NÉZ KI EGY SPRING BOOT APP?

DEPENDENCY INJECTION

DEPENDENCY INJECTION

- Minden olyan osztályt, amelyet elérhetővé szeretnénk tenni a DI számára, el kell látnunk egy @Component annotációval, ezután az adott osztály injektálható @Autowired segítségével
- Vannak kitüntetett @Component-ek, mint a @Service és a @Repository
- @Service - olyan egyedülálló objektum amely elsősorban műveletek elvégzésére szolgál - általában ide szervezzük az üzleti logikát
- @Repository - olyan objektum amely adattárolásra, adatelérésre vagy bármilyen külső kommunikáció reprezentálására szolgál

GRADLE BUILD TOOL

GRADLE BUILD TOOL

- A Gradle egy kód menedzselésére és build folyamatok automatizálásra szolgáló szoftver
- Konfigurációja Groovy-ben történik
- Láttatok már ilyet: npm
- Főként 3. Féltől származó függőségek betöltésére használjuk (FIGYELEM ez nem a DI!)
- Ezen felül nagyon sok plugin támogatását élvezi, amelyek segítségével könnyebbé válhat az életünk - Leandreck, mapstruct

SPRING REST API

REST API

- Ha már tervezési minta, akkor kell nekünk @Controller, de mivel REST API-t akarunk írni, így adjuk meg pontosabban @RestController-ként

- A controllerünkben akarunk endpointokat írni, amelyek lényegében HttpRequest Mappelések, általános esetben @RequestMapping(“/”) -nek nevezzük, viszont ha pontosítani akarunk a metóduson, akkor lehet pl. @GetMapping(“/”), @PostMapping(“/”), @PutMapping(“/”), @DeleteMapping(“/”)

REST API – ADATKÜLDÉS

- @PutMapping("/pathvar/{var_name}/edit")
  public String putMappingMethod(@PathVariable("var_name") String pathVar)
  {return pathVar;}
- @GetMapping("/findAll") // localhost:8080/findAll?search_value=mivan?!
  public String getMappingMethod(@RequestParam("search_value") String searchValue)
  {return searchValue;}
- @PostMapping("/save") // itt fontos a content-type: application/json
  public Map<String, Object> postMappingMethod(@RequestBody Map<String, Object>
  jsonData)
  {return jsonData;}
- // TODO header values

REST API – TESZTELÉS POSTMANBŐL

CORS

ANGULAR ELÉRÉS