**Mi a Spring Framework? (3)**

Java-s keretrendszer **nagyvállalati alkalmazásfejlesztésre**. Keretrendszer, mert az általunk létrehozott **komponenseket ő kezeli** (példányosítja le, állítja be a kapcsolataikat, vezérli az életciklusukat). **Célja**, hogy a **fejlesztő az üzleti logikára koncentrálhasson azáltal,** hogy az ismétlődő, kódigényes stb. dolgokat leveszi a válláról.

**Mi az a nagyvállalati java alkalmazás? (3)**

Olyan alkalmazások, ahol már **nem elégségesek a Java SE-be beépített eszközök/tulajdonságok**, vagy azokat csak túl nagy munka árán lehetne megvalósítani. Ezért fejlesztésükkor **valamilyen keretrendszert alkalmaznak** ezen extrák beépítésére; a keretrendszer **saját és/vagy már létező 3rd party megoldásokat/eszközöket egy egységes modellbe integrálva** szolgálja ki ezeket az igényeket.

**Mik ezek a konkrét igények, tulajdonságok, amiket a nagyvállalati alkalmazások elvárásként támasztanak a fejlesztés felé? (12)**

**komponensek életciklus kezelése és kapcsolatok**

**távoli elérés** – tipikusan weben keresztül

**többszálúság** – a párhuzamos felhasználás biztosított legyen

**perzisztencia** – a JAVA SE-s JDBC elavult és sok benne a boiler template kódolás

**tranzakciókezelés** – atomic(=vagy az összes, vagy egyik sem) műveletek biztosítása

**aszinkron üzenetküldés** – üzenet küldés és fogadás az alkalmazások között (aszinkron: nem várja meg a választ)

**ütemezés** – műveletek lefutása valamiféle ütemezés szerint, ne felhasználói beavatkozásra

**integráció** – más alkalmazásokkal, különböző módokban

**auditálhatóság** – a felhasználó írási műveletei legyenek visszanézhetőek

**konfigurálhatóság** – különböző beállítások lehessenek különböző környezetben való futtatásokra

**biztonság** – bejelentkezés és felhasználó jogainak korlátozása

**tesztelhetőség** – legyen rá lehetőség és legyen automatizált

**A nagyvállalati alkalmazás igényeket a SpringFramework milyen eszközökkel, tulajdonságokkal, elvekkel elégíti ki? (6)**

**Pehelysúlyú komponensek használata** - egyszerű, hétköznapi java objektumok(java beans, pojo)

**Konténer használata** – a komponensek felügyeletét a Spring konténerre, az ún. Applicaiton Contextre bízza

**Dependency Injection** és **Inversion of Control**

**Aspektusorientált programozás támogatása** –bizonyos műveleteket egy közös helyre össze lehet gyűjteni, majd konfigurációval bekapcsolni; ilyen pl. a *security*, *naplózás*, *tranzakciókezelés*

**3rd party libraryk integrációja** – A Framework nem mindent önmaga implementál, hanem beépíti a leggyakoribb legelterjedtebb 3rd party libraryket is

**Boilerplate kódok eliminálása** – fő cél: csak az üzleti logikára kelljen koncentrálni, minden haszontalan, ismétlődő kódírást a keretrendszer vegyen le a vállunkról

**Mi az a Dependency Injection technika és milyen módjai vannak?**

Vagy objektum összekapcsolása másik objektummal =objektum injektálása egy másik objektumba. Dependency Injection-nek nevezzük amikor egy Java osztály típusú attribútumának valamilyen **külső** módon értéket adunk, vagyis a minta alapkövetelménye, hogy **a szolgáltató objektumot nem a kliens hozza létre(nincs new!), hanem külsőleg kapja meg. A kliens nem hívhatja a szolgáltató konstruktorát; a függőséget előállítja valaki más, pl. a konténer!**

A kliens külső kódnak delegálja függőségeinek létrehozását (az injektornak), amit azonban nem hívhat. Az injektor adja oda a szolgáltató objektumot a kliensnek.

Több konstruktor esetén az **@Autowired annotáció** jelzi, melyiket kell használja majd a SpringBoot a DI-re; egy konstruktor esetén nem kell ezt jelölni.

**Mik a Dependency Injection előnyei?**

az objektum független attól, hogy az általa megkövetelt objektumok hogyan jönnek létre

az eredetileg merev kód feloldása - a tartalmazott objektumot kicserélhetővé válik a kliens módosítása nélkül.

több különböző konfiguráció támogatása; minden objektum tulajdonságai külön konfigurációs fájlban

**A Springnek tudnia kell, hogyan példányosítsa le a Controllert az AC-ben tárolt Service példánnyal** -> konstruktor vagy setter injection-nel? Konstruktor injection megadása esetén egyértlemű neki, **setter injectionnél a setter metódusra rá kell rakni az @Autowired annotációt.**

**Nem DI és DI példák**

*Nem dependency inejction:*

**public** **class** **Client** {

**private** ExampleService service;

*// Constructor*

Client() {

service = **new** ExampleService();

}

}

*Dependency inejction*

*// Constructor*

Client(Service service) {

**this**.service = service;

}

*/Setter*

**public** void setService(Service service) {

**this**.service = service;

}

**Hogyan lehet a Servie osztályt magunknak példányosítani?**

@levesszük a Service-t

EmployeeApplicaiton belül példányosítom a HelloServicet egy statikus metódussal

rárakom a metódusra a @Bean annotációt

**Mi jelent a kofngiruáció Springben?**

Amikor a Spring frameworkot non invasive módon használjuk, és nem hivatkozunk rá az adott osztályból, az osztályt nem olvastatjuk fel a Component Scannel azért, mert saját magunk akarjuk megadni(=explicit módon), hogy hogyan történjen az objektum példányosítása. Ugyanakkor azt szeretnénk, ha ez a példány bekerülne az ACbe és a Spring kezelné. Ehhez @Configuraitont teszünk az osztályra, ahol a példányosító metódusunk lesz, és @Beant a példányosító metódusra.

**Mi a @Bean annotáció jelentése?**

Az ilyen annotációval ellátott metódusok visszatérési értékeként kapott példányt a Spring beanként fogja kezelni -> el kell helyeznie az Applicaiton contextbe és ő vezérli, példányosítja, stb.

A saját komponeneseinkre ez nem jó gyakorlat: egy éles alkalmazásbab több 100 Controller és Service lehet. Inkább a 3rd partykra jó az ilyen fajta konfiguráció. Valamilyen konfiguráció mindenképpen kell Springnél a működésre.

**Mi az az Inversion of Control technika?**

Vagy a kontrol megfordítása. Röviden IOC.  Főleg [objektumorientált programozási nyelvekben](https://hu.wikipedia.org/wiki/Objektumorient%C3%A1lt_programoz%C3%A1s) használt technika a komponensek összeillesztésére, konfigurálására és kezelésére.

A technika lényege, hogy a **komponenskezelést** (pl. létrehozást, példányosítást, paraméterezést, megszüntetést, metódus hívás) **kiemeljük** a programkódból, és általában egy külső keretrendszerre bízzuk. Célja ezzel a **modularitás növelése.**

A **hagyományos programozással** szemben, ahol a **főprogram végzi a hívásokat**, itt a vezérlés megfordul, a keretrendszer **irányít**, a **fejlesztő kódja** pedig a **keretrendszer által nyújtott keretet tölti ki**.

AZ IC egyik megvalósulási formája a DI.

**Mik a Spring Frameworkben fejlesztett alkalmazások komponensei, építőkövei?**

a **Spring beanek,** amelyek egyszerű, hétköznapi specifikáció-, interface- és elnevezési függőség nélküli, pehelysúlyú komponensek, JAVA objektumok, POJO-k (plain old java object= jó öreg java objektum).

Ezeket a fejlesztőknek kell implementálnunk, de kezelésük, vezérlésük a konténer feladata.

Alapesetben **minden bean csak egyszer jön létre** (**singleton tervezési minta**) -> egy példány jön csak létre; **ha bejön egy webes alkalmazás esetén egy http kérés, akkor ugyanez a példány kerül megszólításra;** ha **tárol valami állapotot, párhuzamossági problémák** keletkezhetnek, ezért vigyáznunk kell erre, kivédhető azzal, ha **nem definiálunk benne olyan attribútumot, ami állapotot tárol**

**Mi a Springes konténer és miért felel?**

A komponensek a Springes IOC konténer felügyelete alatt állnak. a Spring IoC DI-t használ az ioC eléréséhez.

A konténert a **BeanFactory** és **ApplicaitonContext** interfészek teszik ki. A BeanFactory a root interfész, amin keresztül biztosított a konténer elérése, alap funkciókkal.

AZ Applicaiton Context(=AC) a BeanFactory leszármazottja sok specifikus vállalati funkcióval, ezért ezt használják defaultként.

**Az AC tartalmazza/gyűjti össze a fejlesztő által implementált komponenseket; felelős a beanek példányosításáért, konfigurálásért, kapcsolataik kialakításáért és az életciklusuk vezérléséért.**

**Mi az az MVC?**

**Programtervezési mintha; az adathoz(modell) és a felhasználói felülethez(nézet) tartozó dolgok szétválasztását szolgálja.** Cél, hogy a felhasználói felület **ne befolyásolja** az adatkezelést, és **az adatok átszervezhetők legyenek a felhasználói felület változtatása nélkül.**

A modell-nézet-vezérlő ezt úgy éri el, hogy **elkülöníti** az adatok elérését és az üzleti logikát az adatok megjelenítésétől és a felhasználói interakciótól egy közbülső összetevő, a **vezérlő** **bevezetésével**.

**Mi az a Spring MVC?**

**A Spring saját webes keretrendszere,** webes alkalmazások fejlesztésére (egy Web MVC Framework) - A Springes fejlesztők megfelelő 3rd party hiányában fejlesztettek egy sajátot. A Model View Controller tervezési mintát követi. Http orientált. Használatra kész eszközöket tartalmaz web alkalmazások fejlesztésére(míg pl. a SpringBoot default konfigurációkat tartalmaz).

Model – üzleti logika; Javaban ezeket a POJO-k reprezentálják

View – megjelenítés

Controller – az alkalmazás vezérlése; Javaban servlet objektumokkal

**Mi az a háromrétegű webes alkalmazás, és hogyan épül fel?**

**Mi az a SpringBoot?**

A Spring Framework kiterjesztése, ami **eliminálja a boilerplate konfigurációt** a **Springes alkalmazás létrehozásakor**. A beanek testreszabásának, összekapcsolásának és a 3rd party libraryk integrálásának terhét leveszi a vállunkról. A **component scannek** köszönhetően **felolvassa és kezeli a beaneket,** az **autoconfiguraitonnek** köszönhetően pedig a classpathon lévő **3rd party libraryket felkonfigurálja** a leggyakrabban használt, **intelligens alapértékekkel**.

**Mik az a SpringBoot előnyei? (5)**

**self contained alkalmazásokat hozhatunk vele létre** – webes alkalmazás esetén nem kell nekünk külön telepíteni webkonténert és arra feltenni a projektünk war állományát, hanem a létrehozott jar állományba be lesz csomagolva a default webkonténer, a Tomcat is

**nagyvállalati üzemeltethetőség biztosítása** – részei az ún. **actuatorok**, amelyekkel az üzemeltetők számára az **infó kinyerés és monitorozás** egyszerűsödik?

**start.spring.io weboldal – le lehet vele generálni egy indító alkalmazás sablont**

**tartalmaz Starter projekteket – pl.** webes alkalmazáshoz Maven dependency esetén elég csak a spring-boot-starter-web-et felvenni; van ugyanilyen pl jpa-ra is

**képes a web konténert is becsomagolni jar állományba** és ezzel elindítható az alkalmazás sajátgépről, mint egy egyszerű javas alkalmazás, nem pedig mint egy webes

**Mi történik a Hellospringboot alkalmazás IDEA-ból történő indításakor?**

elindul a **build**, fordításra kerülnek a javas osztályok; létrejön a jar állomány, amibe becsomagolódik a webkonténer is

**majd elindításra kerül a SpringBootos alkalmazás a beépített Tomcat containerrel**; telepítésre kerül a webkontérbe az alkalmazás

az **alkalmazás a 8080as proton** (sajátgép, mint web server host) **hallgatózik**, és várja a kéréseket

A böngésző frissítésével beérkezik a http request, és dinamikusan frissül a kiírt szöveg

**Hogy néz ki a start.spring.io által legenerált template Maven esetén?**

*struktúra, állományok*

egy .zip állomány, amit ki kell csomagolni és megnyitni a fejlesztő eszközben

klasszikus maven struktúra pom.xml-el

kiemelt fontosságú állomány az **applicaiton.properties**, a bootal kapcsolatosan minden itt van konfigurálva; eredetileg üres?

statics könyvtár: webes alkalmazások esetén ide kerülnek a nem módosuló statikus html oldalak, css és javascript fileok

templates könyvtár: ha server oldali html generálást akarunk, az adott template ehhez ide kerül

*pom.xml*

a file egy **másik projektnek lesz a leszármazottja** -> egy **parent project** lesz definiálva, ahol fel vannak véve a **függőségek verziószámai** (org.springframework.boot::spring-boot-starter-parent)

**webes alkalmazás** kiválasztása **esetén** felveszi függőségként a **default konténert a Tomcatet,** és felveszi a **Jacksont**, ami a **Java objektumok** és a **jSON dokumentumok** közötti **megfeleltetésért** feleős

felveszi a **Hibernate Validatort** is, amivel a bean validaiton API-t tudjuk használni, a **beérkező adatok validálására**

**Teszthez szükség van a boot starter-test moduljára**; ez sok függőséget felvesz; ma már a jUnit5öt használja; mockolásra a Mockitot; assertelésre a Hamcrestet és az Assertjt; használjuk!); XML összrhasonlításra XMLUnit, de elterjettebb formátum REST esetén a jSON, ehhez az összehasonlításra JSONassert, és json kifejezéske írására a JsonPath

*osztályok*

*két előre legenerált osztály:*

**XXXApllicaiton.java**

ezzel lehet elindítani az alkalmazást

@SpringBootApplication annotáció van rajta

az alkalmazás belépési pontja, a main() metódus is itt helyezkedik el

**XXXTests.java** -> egy üres tesztesetet tartalmaz, ami a Spring jó konfigurálását és elindulását vizsgálja

**Milyen beállításokért felel a @SpringBootApplication annotáció?**

*egy metaannotáció, ő is több annotációt tartalmaz*

**@SpringBootConfiguraiton**

amin ez rajta van abban Java configot el lehet helyezni

**@EnableAutoConfiguraiton**

a classpathon lévő 3rd party libraryket fel fogja konfigurálni intelligens alapértékekkel; helyette az explicit konfigurációhoz ez kell: **@SpringBootConfiguration -> @Configuration**

**@ComponentScan -** automatikusan felolvassa/felderíti a csomagban lévő osztályokat és beanként kezeli (**példányosít, életciklust vezérel és a függőségeket DI-vel kezeli**) majd azokat, amelyeken rajta vannak a következő annotációk: **@Component**(általános objektumok), **@Repository, @Service, @Controller**

**Mire jó az annotáció és mikhez rendelhető hozzá Java-ban?**

A [Java programozási nyelvben](https://hu.wikipedia.org/wiki/Java_(programoz%C3%A1si_nyelv)) annotációnak olyan **nyelvi elemet** nevezünk, amely lehetővé teszi a **metaadatok**(=adat az adatról) **beépítését forráskódba.**

Hozzárendelhető csomagokhoz, osztályokhoz, metódusokhoz, változókhoz, paraméterekhez.

**Mi a Controller?**

Spring MVC komponens, bean. Feladatuk, hogy a **felhasználótól a bejövő kéréseket értelmezzék, és választ adjanak neki vissza**. Ha bejön **egy http request, a Spring ezt átalakítja metódus hívássá** és a metódus **visszatérési értékét** kiírja a http **responseba.**

Sima POJO, @Controller annotációval. Az annotációnak köszönhetően a component scan felolvassa, java beanként értelmezi és beteszi egy példányát az AC-be.

**Milyen további Controller annotációkat ismersz?**

**@RequestMapping(”XXX”)**

Megadatjuk vele, hogy milyen url-en figyeljen az adott metódus (default beállítás: localhost:8080). Meg lehet adni nem csak metódus, hanem osztály szinten is, ilyenkor a összes metódusra ez vonatkozik. Megadható wildcard karakterekkel is (pl ..\*.html)

**@ResponseBody**

Ezzel a beállítással a metódus a visszatérési értéket valamilyen formátumba lemappeli (default a jSON) és http responsként ez továbbítódik.

**Miért fele a @RestController annotáció?**

Metaannotáció ami magába foglalja a @Controller és a @ResponseBody annotációkat. Használatával nem kell minden controller metódus elé kiírni a @ResponseBody-t.

**Mit tudsz a springes beanek kapcsolatairól, mik a jellemzői? Az AC a kapcsolatok beállításakor hogy szolgálja az IC mintát?**

Tipikusan a magasabb szintű komponens lát rá az alacsonyabb szintűre. A Controller csak a Servicet látja, a Service csak a Repositoryt; A beanek látják a maguk szintjén lévő többi beant iss (pl. egy service egy másik servicet), a lényeg csak, hogy A repo-service-controller modellben fölfelé ne legyen függőség.

A laza kapcsolat, a modularitás jegyében a függőségeket a Spring dependency injection-el állítja be -> Az AC-ben tárolt és lepéldányosított Service-t injektálja a controllerbe, amikor példányosítja), amivel kiszolgálja az inversion of control elvet. A Spring a példányosyításoknál tudni fogja a sorrendet, a függőségektől függően.

**Mikor kell Java konfiguráció Springes projekt esetén és mit jelent?**

Amikor a Springet non invasive módon használjuk; ilyenkor a Springe sprojekten belül bizonyos osztályainkban nem hivatkozunk a Springre, mert nem akarjuk, hogy a component scan mechaizmussal felolvassa őket, mert pl. mi akarjuk explicit módon megadni, hogyan plédányosítsa majd le az adott beant. Ehhez @Configuration-el kell az osztály ellátni, és kell bele egy példányosító metódus, @Bean annotációval. Ennek hatására a Spring ezt a metódust meghívja és a visszatérési értékét elhelyezi az AC-ben. Legjobb gyarkolat szertrint viszont a saját komponenseinket a componenscan mechanizmusra bízzuk, és csak 3rd partykat látunk el java konfigurációval.

**Mi az a Maven wrapper?**

**A SpringBootos alkalmazások tartalmazzák, ez egy script, ami letölti a Mavent**. Segítségével akkor is tudjuk a Maven parancsait használni, ha nincs a gépre telepítve: megfelelő paranccsal indítva az alkalmazás részét képező scritpt először letölti a Mavent, hogy azután lehessen használni.

mwnw clean package – jar állomány létrehozása

mvnw spring-boot:run – futtasás ( =java – jar xxx.jar)

**Mi történik a jar állománnyal ha SpringBootos projectet buildelünk?**

SringBootos alkalmazás esetén a pom-xmlbne lévő spring-boot-maven-plugin hatására, ha kiadjuk a mvn clean package parancsot, a build során a **létehozott .jar állományt átcsomagolja úgy, hogy magát a webkonténert(Tomcat) is belecsomagolja**. **Az eredeti -jar állományt átnevezi jar.original-re, és létrejön egy új nagy jar állomány a Tomcat containerrel**.

**SpringBootos project indításánál defaultként hogyan települ Tomcat és mit csinál? Hogyan érhető el?**

A Tomcat server(=web konténer) a sajátgépünkre települ. A server konfiguráció során beállítja a protocollt(http), a portot(8080) és még sok minden mást.

A települt webkonténeren keresztül egy tiszta java http web server környezet jön létre, amibe automatikusan feltelepítésre kerül a Java alkalmazásunk és el is indítódik. **A konténer, mint server elérhetővé teszi a benne lévő tartalmat a hálózaton.**

A konténert és rajta keresztül az alkalmazást a web böngészőbe írt a [http://localhost](http://localhost/):8080as címen fogjuk elérni, ahol a futó Java alkalmazásunk figyel és várja a beérkező http requesteket.

a Tomcat kofigurációs filejai XML formátumban a  c:\myWebProject\tomcat\conf mappában helyezkednek el.

**Hogyan unit tesztelünk Springes projekt esetén?**

Unit teszt az, amikor **csak egy java osztályt tesztelünk le, a függőségei nélkül.** Ilyenkor a **Springet non invasive** módon használjuk, ezekre a tesztekre nem indítjuk be. Ha az osztálynak nincs függősége más objektumra a unit tesztelést a default módszerrel simán megtehetjük. **Függőség esetén** a service-t **kicseréljük egy test doublere**, egy **mock objektumra** és használjuk az **@Extendwith(MockitoExtension.Class),** a @**Mock** és az @**InjectMocks** annotációkat.

**Hogy írunk Spring Boots project esetén integrációs teszteket?**

Az adott teszt osztályt a **@SpringBootTest** annotációval kell ellátni. a Spring a különböző tesztesetek közötti cache-eli az application contextet, vagyis az összes teszt ugyanabban az contextben fog lefutni, ezért vigyázni kell arra, hogy **ne legyen az AC-nek állapotváltozása**. A beanek a tesztbe az **@Autowired** annotációval injektálhatóak. A tesztosztály nevét integrációs tesztek esetén **’IT’ végződéssel** látjuk el; ezzel a Mavennek nem a surefire pluginja, hanem a **failsave pluginja fogja elindítani és ezzel** külön tudjuk választani, hogy melyik fázisában melyik tesztesetek fussanak le(unit és integráiós tesztek külön).

**Mi a Docker?**

**A Docker egy OP rendszer szintű virtualizációt valósít meg, aminek hatására különálló, jól elkülönített környezeteket tudunk létrehozni saját filerendszerrel és saját telepített szoftverekkel**. A **klasszikus virtualizációtól** annyiban eltér, hogy **kevésbé erőforrás igényes, nem kell a teljes op rendszert újra telepíteni,** minden egyes részét újra futtatni, hanem a host op rendszernek a szolgáltatásait tudjuk igénybe venni.

**Hogy működik egy SpringBootos alkalmazás Docker környezetben?**

create dokcerfile -> run mvn clean package -> create image -> run container

A SpringBoot beépítetten tartalmaz Docker támogatást. Amikor Mavennel lebuildeljük az alkalmazást, az becsomagolódik egy .jar fileba a Tomcat web konténerrel együtt. Ezt a jar filet, az összes, az alkalmazás futásához szükséges egyéb dologgal (op rendszer, jdk környezet) bele kell becsomagoljuk egy docker image-be. a docker imaget a dockerfile-ban definiáljuk, amit a projekt főkönyvtárában helyezünk el. a Fileban megadjuk, hogy már milyen meglévő környezetből indulunk ki(FROM), majd RUN és COPY parancsokkal átmásoljuk a jar állományt a targetből a konténerbe, végül a CMD (tömbösen!) paranccsal elindítjuk az alkalmazást.

Az image létrehozása előtt futtassunk egy cleqan packaget, hogy a legfrisebb verziót legyne majd az imageben. Az imaget a docker build -t(=tag) név .(=ebben a könyvtárban)

Magát a konténert az imageből a docker run -al tudjuk elindítani, amiben futni fog az alkalmazásunk. A Dockerben fut a Tomcat webserver, amin keresztül az alkalmazással docker konténer a 8080as portján keresztül lehet kommunikálni. Ezt ki kell vezetni a sajátgép 8080as portjára úgy, hogy a docker run parancs kiadásakor -p kapcsolóval megadjuk ezt a kivezetést(honnan hova: 8080:8080 ).

**Milyen Docker parancsokat, kapcsolókat ismersz?**

docker build -t employees . –> készítse el az imaget a könyvtárban található docker file alapján és adja az imagenek az employees nevet

docker images – kilistázza az imageket

docker run -d -p 8080:8080 –name my-employees enmployees – elindítunk egy konténert az employees imageből úgy, hogy a konténer 8080as portját, amin kereszütl az alkalmazás a hálózaton elérhető, kivezetjük a sajátgépünk 8080as portjára; a konténerhez a my-employees nevet rendlejük hozzá; -d-vel visszakérjük a portot

docker logs -f – a docker log folyamatos kiírása, elveszi a promptot

ctrl c – logolás megszakítása, prompt vissza

docker ps – futó docker konténerek kilistázása

docker stop my-employees - -a my-employees konténer leállítása

docker ps a – futó és nem futó docker konténerek kilistázása

docker start my-employees – konténer újraindítása

dokcer rm container – konténer törlése

docker rmi image – image törlése

**Ha nem a Docker parancsaival, hanem a Mavenből akarod a Dockert vezérelni, milyen parancsokat kell használni?**

mvn package docker: build – legyártja az imaget

mvn docker: start

mvn docker: stop **- töröl is!**

**Milyen előnyei vannak a Docker használatának a fejlesztés során?**

**Ha fejlesztünk, a saját gépes környezet eltérhet az éles környezettől, így** az alkalmazásunk másképpen is működhet. Docker használatával kiküszöbölhető, hogy előjöjjenek olyan hibák éles környezetben, amik a fejlesztés során nem.

**Egyszerre több alkalmazás fejlesztésekor** több ilyen környezetet is tudunk egyszerre használni, amik az adott környezetben futnak függetlenül a sajátgépünktől, **nem kell mindent feltelepítünk egy helyre.**

A többféle környezet bevonásával **több számítógépből álló hálózatot is le tudunk modellezni – pél. az egyik docker containerből elindítunk egy adatbázist, másikból egy alkalmazást stb.**

microservice-ek esetén különösen fontos lehet a docker használat -> kész környezetet kapunk, a megfelelő működéssel -> könnyű, gyors

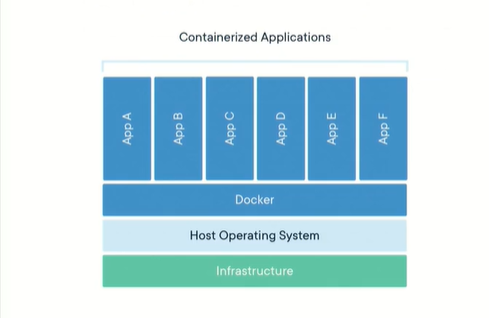
**Milyen egyéb tényezők ösztönzik a Docker használatát?**

üzemeltetők, tesztelők szempontjai; **devops** irányzat

**Hogy néz ki a Docker architektúra?**

**Az infrastruktúra (fejlesztés közben a sajátgép, éles környezet esetén a webserver)van alul, arra telepítjük az op rendszert(=host op rendszer) és arra a Dockert.**

A Docker különböző környezeteit/konténereket amikor futtatjuk, ezeket mind ez a host op rendszer szolgálja ki, nem kellenek külön



**Mi az a Restful webszolgáltatás?**

**2 stack commsára integrációs tech; alkalmazás adatai = erőforrások; erőforrások hozzáférése urlen keresztül; 4 alapművelet engedélyezett az erőforrásokon http metódusokon keresztül; visszaadott adatok szerializációját is előírja, ami tipikusan jSON**

**megvalósítás: Controller osztály a kliens requestek feldolgozására, kiszolgálása: rest végpontok létrehozása =urlek amikhez megadjuk, hogy milyen beküldött http metódusra milyen javas metódusok legyen meghívva; a visszaétrési értékeket szerializáljuk (=@Responsebody)**

**(erőforrások urlel történő azonosítása, szerializáció-jSON)**

**A két stack(backend és frontend) kommunikációját biztosító integrációs technológia.**

REST = Representational state transfer.

A REST egy architektúra mintha. Az alkalmazásokra úgy tekint, mint **erőforrások** gyűjteményére.

**REST** esetén az alkalmazásban található **erőforrásokat az url-ek azonosítják**.

A Restul web alkalmazások a **metódusok visszatérési értékét(az erőforrásokat)** **szerializálják** és a frontend számára **url-ek hozzárendelésével**(= címzés, egyedi azonosítóval ellátás) **teszik elérhetővé**. Példa <http://demo.guru99.com> és http://demo.guru99.com/employee/1 . Mit akarunk csinálni a resourccel? Itt jönnek be a http metódusok.

**A szerializáció leggyakrabban JSON formátumban történik és célja, hogy a különböző programnyelvek által generált, vagy különböző formátumú adatokat egységessé tegye**. A szerializált erőforrásokkal az **egyszerűséget, skálázhatóságot és platformfüggetlenséget** szolgálja.

**Springbe hogyan kell rest webszogáltatásokat implementálni?**

Spring MVC-t kell használni. Controller osztályban @RestController(=@ & ResponseBody minden metódusra) Controller, @\*\*Mapping annotációk használatával

**Mi az a REST és a SOAP közötti különbség?**

a SOAP egy protokoll (=Simple Object Access Protocol) a REST egy architektúra. A SOAP csak XML formátummal működik, míg a REST XML, HTML és jSON formátumokat is támogat. A REST a SOAP hibát küszöböli ki; egyszerűbb, több formátumot támogat

**A Restful az erőforrásokon milyen műveleteket enged csak elvégezni és ehhez milyen metódusokat használ?**

Az erőforrásokon csak a 4 alapvető műveletet engedi elvégezni > CRUD (create, read, update, delete). Ezeket a műveleteket http metódusokon keresztül végzi el (PUT, GET, POST, DELETE).

.

**Mi a GET művelet?**

**Erőforrás lekérése szolgál,** CRUD megfelelője a READ. Ha a kliens webböngészője http get metódussal éri el a webservert(=http request), egy lekérdezés hajtódik végre, ami az urlnek megfelelően alkalmazás Controllerében beállított metódus visszatérési értékét kérdezi le, és a defaultnak megfelelően **jSON formátumban visszaküldi a responsebodyba** (=http response).

**Mit jelent a @GetMapping?**

Az ezen az annotációval ellátott java-s metódus a megadott urlen lesz elérhető, http GET metódussal.

**Mi az a DTO?**

Osztály, ami az **eredeti entitás állapotát tárolja csak**, üzleti logikát nem, az eredeti **metódusaihoz nem lehet hozzáférni** (DAO-tól eben különbözik). Ennek **biztonsági** és **olvashatósági okai** vannak.

**A befelé jövő DTO-k a commandok** -> **ezeknél lesz állapotváltozás** -> INSERT vagy UPDATE művelet lesz az adatbázisban

Az entitások és a DTO-ok között kelleni fog konvertálás. Egyik eszköz erre a Modelmapper.

**Hogyan működik a Modelmapper?**

Létre kell hozzunk egy DTO osztályt az entitás osztály atrijaival. Példányosítanunk kell egy Modelmappert, amit előzőleg függőségként fel kell venni a pom.xmlbe. A modellMpper.map() metódusával az entitást DTO-vá konvertáljuk.

Listáknál a konvertáláshoz a Typeot ( Type targetListType = new TypeToken<List<EmployeeDto>>() {}.getType(); objektumnál pedig xxxDTO.classként kell megadni.

**Hogyan paraméterezünk GET műveleteket és ez mire jó?**

**prefix érték átadásával az URLben** -@**RequestParam** annotációval látjuk el a metódusunk aktuális paraméterét; az urlből kiszedi a prefixet és azt adja tovább a lekérdező service metódusnak

vagy *URL részletének kezelése (*pl. ID-val) ->**@PathVariable-el**  definiáljuk az aktuális paramétert; a **@GetMappingnek placeholderrel** kell átadni pl. az id-t

**Mi az az idempotens művelet?**

Amikor az adott művelet többszöri elvégzésre is ugyanazt az eredményt adja vissza. Ilyen a http metódusok közül a POST. A böngészők csak a POST metódust ismerik, ezért kérdeznek rá mindig újraküldéskor, hogy biztos?

**Mi az általános különbség a create és update műveleteknél, és melyikre melyik http metódust használjuk?**

A create-nél van id küldés is, ilyenkor a POST művelet kell. Updatenél nincs ID, ilyenkor az idempotens PUT kell.

**A http metódusokkal elérhető Controller metódusoknak mi a visszatérési értéke?**

A GET, POST és PUT műveletnek DTO, a DELETE metódushoz rendelt java-s metódusnál nem adunk vissza semmit.

**Hogyan hozunk létre http GET metódussal elérhető java-s metódust a Controllerben?**

A @GetMapping annotációval. Jelentése: a megadott urlen a java-s metódus elérhető http GET metódussal (= a kliens webböngészője a megadott urlen a webservernek http HET requestet küld). A metódus paraméterezhető a @RequestParam-ként átadott, Optinalként kezelt, az url-ből Stringként kiszedett prefixel, vagy ha a @GetMappingnek placeholderként adjuk meg az idt, amit az urlből kiszedve @PathVariable-ként adunk át a metódusnak. Mindkettőnél működik a típuskonverzió (Stringből Optional<String> és Stringből long lesz). A prefixes megoldást szűrésre használjuk, a @Pathvariable-s megoldást pedig akkor, ha egy konkrét objektum kell.

**Hogyan hozunk létre http POST metódussal elérhető java-s metódust a Controllerben?**

A POST idempotens művelet, vagyis többszöri hívásra ugyanúgy végbemegy. Ha az urlben nem szerepel az id és a http request egy POST(@PostMapping), akkor legyen elérhető a java-s metódus. A metódusnak egy paramétere lesz: @RequestBody CreateXXXXCommand, jelentése: A @RequestBodyból kell kinyernie és deszerializálnia (=jsSON formátumból CreateXXXXCommand formátumba) az adatot. Ezzel a commanddal hívja meg a controller a service megfelelő create metódusát, amely a command és egy AtomicLong típusú idGenerator alapján lepéldányosít egy tárolandó objektumot és eltárolja. Miután végbemegy, visszatér DTO-val.

**Hogyan hozunk létre http PUT metódussal elérhető java-s metódust a Controllerben?**

A PUT nem idempotens művelet, csak egyszer hajtódik végre. Ha az urlben szerepel az id (@PutMapping(”{id}”) is, és a http request egy PUT, akkor legyen elérhető a java-s metódus. A metódusnak két paraméterre is szüksége lesz: @RequestBody UpdateXXXXCommand és @PathVariable (”id”) long id; jelentése: A @RequestBodyból kell kinyernie és deszerializálnia UpdateXXXXCommand-á az adatot. Ezzel az id-val és commanddal hívja meg a controller a service megfelelő metódusát, ami elvégzi az updatet és visszatér DTO-val.

**Hogyan hozunk létre http DELETE metódussal elérhető java-s metódust a Controllerben?**

Ha az urlben szerepel az id és a http request egy DELETE (@DeleteMapping), akkor legyen elérhető a java-s metódus. A metódusnak egy paramétere lesz, @PathVariable-ként kapja meg a törlendő objektum id-ját. . Ezzel az id-val hívja meg a controller a service megfelelő metódusát, ami elvégzi a deletet és void a visszatérése.

**Mi az a Command?**

**Olyan DTO, ami majd módosítja az alkalmazás állapotát;** a bejövő POST, PUT http requestek kezelésére hozzuk létre.

**Mi az a deszerializáció?**

A szerializáció ellentétje; a Commandként beérkezett adat átkonvertálása jsonból Jacksonnel DTOvá

**Mért felel a @RequestBody annotáció?**

Ahhoz, hogy tudja a Spring, hogy a ReuestBodyból kell kinyerni az adatokat a commandhoz jSON alapján, @RequestBodyt használunk a metódus aktuális command paramétere előtt.

**Hogyan generálhatunk http request metódusokat, hogy lecsekkolhassuk az alkalmazásunk működését?**

Az adott url betöltésekor a webböngésző automatikusan GET http requestet küld a webservernek; azonban POST és DELETE műveleteket nem tudunk a böngészővel eszközölni.

**IDEA-ban test ágon hozzunk létre új http requestet**. Add requesttel tudunk hozzáadni; beállítjuk az urlt. Az alkalmazás indítása után a zöld nyíllal tudom meghívni a metódust.

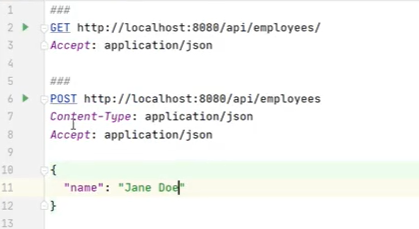
**Mit tartalmaz a http request?**

Metódus neve és hogy milyen urlen legyen beküldve

Content type amit requestként küld (ha van content; update vagy createnél)

Accept type – amit responseként vár

A CreateCommandnak megfelelő jSON beíársa {} között, Stringekkel; = REQUESTBODY!

****

**C:\Users\djtes\AppData\Roaming\JetBrains\IdeaIC2020.3\settings.zip**

**Mi az a web szerver?**

**Hálózati kommunikáció** eszköze; a webszerver olyan **számítógépes szoftver és mögöttes hardver**, amely HTTP-n, a weboldalak terjesztésére létrehozott hálózati protokollon vagy a HTTPS biztonságos változatán keresztül fogadja **el a kéréseket**. **A kliens webböngészőn keresztül** a kommunikációt úgy **kezdeményezi**, hogy egy HTTP-n keresztül igényel egy adott erőforrást, **a szerver pedig az erőforrás tartalmával vagy egy hibaüzenettel válaszol**.

A webkiszolgálóról küldött erőforrás lehet a kiszolgáló számára elérhető, **már létező fájl** , vagy a kérés időpontjában **egy másik, a szerverprogrammal kommunikáló program is létrehozhatja.** Az előbbi gyakran gyorsabb és könnyebben tárolható gyorsítótárba az ismételt kérésekhez, míg az utóbbi az alkalmazások szélesebb körét támogatja. A generált tartalmat kiszolgáló webhelyek általában tárolt fájlokat tartalmaznak, amikor csak lehetséges.

**Mi az a webkonténer vagy servlet container?**

**A**[**webszerver**](https://hu.wikipedia.org/wiki/Webszerver)**azon komponense,** amely a **servleteket kezeli.** A container dolga a **servletek életciklusainak** a kezelése és az [**URL**](https://hu.wikipedia.org/wiki/URL)**-ek hozzárendelése** a servletekhez. Ilyen servlet container például a **Tomcat** vagy a **Jetty**.

web server(Apacha Tomcat) - > webkonténer

web server content can be reached via ursl with a wb client.

web client = böngésző; requestet küld a servernek, hogy kiolvashassa az urlen lévő tartalmat és megjelenítse a felhasználó számára

a web servernek és a web kliensnek közös nyelv kell a kommunikációra, ez a html, és protocolll, ez a http, ami a TCP/IP-re épül. A böngésző http requesteket küld a server felé és http Responseokat fogad.

**Tomcat** provides a "pure Java" HTTP web **server** environment in which Java code can run.

The context **path** refers to the **location** relative to the server's address which represents the name of the web **application**. By **default**, **Tomcat** derives it from the name of the deployed war-**file**. So if we deploy a **file** ExampleApp. war, it will be available at <http://localhost:8080/ExampleApp>,

**Mi az a servlet?**

**Egy olyan**[**Java**](https://hu.wikipedia.org/wiki/Java_(programoz%C3%A1si_nyelv))[**objektum**](https://hu.wikipedia.org/wiki/Objektum)**, amely**[**HTTP**](https://hu.wikipedia.org/wiki/HTTP)**kérést dolgozik fel és HTTP választ generál.** Ezzel a **dinamikus tartalomgenerálás** problémáját oldja meg. **Java Servlet** API tartalmazza a servlet és a servlet container kommunikációjához szükséges osztályokat. A servletek a webalkalmazás többi részével együtt **bekerül a**[**WAR fájlba**](https://hu.wikipedia.org/wiki/WAR_(f%C3%A1jlform%C3%A1tum)).

**Mi az a war file?**

**Mi az a servlet életciklus?**

1. A container **példányosítja** (létrehozza) a servlet objektumot.
2. A container meghívja a servlet példány **init()**metódusát. Ez a metódus inicializálja a servletet és mindenképp le kell futnia **mielőtt a servlet HTTP kéréseket** tudna fogadni. Az init() metódus csak egyszer fut le a servlet élete során.
3. Az inicializációt követően a servlet **képes a klienseket kiszolgálni**. A container minden HTTP kérésre meghívja a **servlet service()** metódusát. Minden kérés **külön szálban** hajtódik végre.
4. A servlet életének záróakkordja az, amikor a container meghívja a **destroy**() metódusát. Az init() metódushoz hasonlóan a destroy() is csak egyszer hajtódik végre a servlet életében.

**Mi az a cachelés?**

= gyorsítótárazás

A **Cache  olyan nagyon gyors mûködésû tároló, amelyben a gyakran használni kívánt adatokat átmenetileg tároljuk**. A számítógép architektúrában a processzor és a memória között van.  Cacheléssel **gyorsabbá** tehetjük honlapjainkat és **gazdaságosabban** üzemeltethetjük a szerverünket. Szinte az összes **tárhelyszolgáltató használ különféle cache megoldásokat**. Lehet vele beállítási problem, pl. a kép módosul de a neve nem. A gyorsítótárazás mindkét esetben az ügyféloldali és a kiszolgálóoldali is végrehajtható

**Mi az az url?**

**Az**[**interneten**](https://hu.wikipedia.org/wiki/Internet)**megtalálható erőforrások szabványosított címe. Egyetlen címben összefoglalja a dokumentum/erőforrás megtalálásához szükséges négy alapvető információt:**

* **a protokollt, amit a célgéppel való kommunikációhoz használunk;**
* **a szóban forgó gép vagy tartomány nevét;**
* **a hálózati port számát, amin az igényelt szolgáltatás elérhető a célgépen;**
* **a fájlhoz vezető elérési utat a célgépen belül.**

A portot gyakran elhagyhatjuk, mert a protokolloknak van alapértelmezett portszáma, a httpnek a 80as, httpsnek a 443-as.

**Mi az az elosztott alkalmazás?**

**Mi az a node?**

= csomópont.

**A hálózat részét képező különálló egység.** A csomópont egy számítógép éppen úgy lehet, mint egy [útválasztó](https://pcforum.hu/szotar/%C3%BAtv%C3%A1laszt%C3%B3) vagy egy [hálózati kártyá](https://pcforum.hu/szotar/h%C3%A1l%C3%B3zati+k%C3%A1rtya)val rendelkező nyomtató vagy más [hardver](https://pcforum.hu/szotar/hardver)[eszköz](https://pcforum.hu/szotar/eszk%C3%B6z).

**Mik azok a threadek?**

szál, vagy **kiszolgáló szál.**

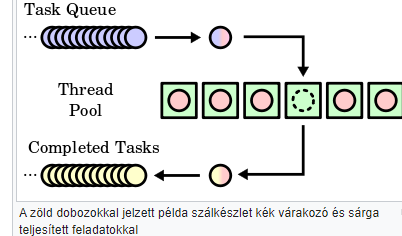
program futásakor a program utasítássorozata eredendően egymásutáni, szekvenciális sorrendben hajtódik végre (=egy folyamat); ezzel szemben párhuzamosan futó vezérlési szálak esetén a folyamat több résre oszlik amik egyszerre futnak.

Egy [programtervezési minta](https://hu.wikipedia.org/wiki/Programtervez%C3%A9si_minta), amely során a szálak azért jönnek létre, hogy párhuzamosan végrehajtsák a meghatározott számú [sorba](https://hu.wikipedia.org/wiki/Sor_(adatszerkezet)) rendezett feladatokat.

Általában több feladat van, mint ahány szál. Folyamatosan érkezhetnek újabb feladatok, amelyek egy sorba kerülnek. A feladatok sora szinkronizált, és lehet elsőbbségi sor is. Amint egy szál elvégzi a feladatát, az kérvényez egy másik feladatot a sorból, amíg minden feladat el nem fogy. A szál ezt követően megszakad, vagy alvó állapotba lép, amíg nem keletkezik új elvégezhető feladat.

**A webszerverek is előszeretettel használják a szálakatFogalom magyarázata:Önállóan működő programegység.(Thread) a böngészőprogramok hatékonyabb kiszolgálására.**

Példa: Quick sortolás esetén a két rész párhuzamos rendezése



**Mi az a pingelés?**

más számítógépek hálózati elérhetőségének ellenőrzésére szolgáló parancs

**Mi az a localhost?**

[**számítógép-hálózatoknál**](https://hu.wikipedia.org/wiki/Sz%C3%A1m%C3%ADt%C3%B3g%C3%A9p-h%C3%A1l%C3%B3zat)**az egyes állomások saját magukra mutató neve.**

Szoftver-fejlesztési célra is használják: a programozó csak egy számítógépet használ, mégis tud kliens- és szerveroldali programokat is írni. **Ha a web böngészőbe beírjuk a**[**http://localhost**](http://localhost/)**címet, automatikusan a saját gépünkön lévő**[**webszerver**](https://hu.wikipedia.org/wiki/Szerver)**kezdőlapját próbálja lekérni.** Ha nem futtatunk http servert, akkor a kapcsolódás sikertelen.

localhost ip címes elérése:

sajátgépről: http://127.0.0.1/

más gépről: <http://192.168.X.X> (xx=local ip)

**Hálózati kommunikáció**

Milliárdnyi server-kliens kapcsolat vezetékes és vezeték és vezeték nélküli hálózatokon keresztül. A kliensek kéréseket(request) küldenek a serverek felé>; a szervereket fogadják ezeket, begyűjtik a válaszhoz szükséges infókat, erőforrásokat és kiszolgálják a klienst, válaszokkal(=response). A válasz rendszerint valamilyen típusú tartalmat jelent. Az elküldött kérést és a visszakapott tartalmat a kliens web böngészőn keresztül küldi/fogadja/jeleníti meg. a tartalom rendszerint HTML formátumban íródik, amit a böngészők képesek megjeleníteni.

Egy weboldal statikus fileok – HTML oldalak, képek sb – gyűjteménye. A webes alkalmazás egy olyan weboldal, ami dinamikus funckionalításokkal rendelkezik server oldalon.

**A http egy protokoll webes server és kliens kommunikációra**; **csak egy kérést per kapcsolatot támogat, azután megszakad** -> **ezzel biztosítja** **több felhasználó kapcsolódási lehetőségét.**

Http kérést GET- és POST metódusokkal lehet létrehozni.

Java-ban a http kérések és válaszok kezelését a servlet objektumok szolgálják.

**Mi az a port?**

Portnak nevezzük azt a bemeneti-kimeneti „**kaput**”, amelyen keresztül **eszközök és hálózatok adatátviteli kommunikációja történhet az adott számítógéppel. Egy azonosítóval rendelkező virtuális cím, ahol a sever és a kliens találkozik és kommunikál, adatot cserél.**

A portok egy hálózati kommunikációs csatorna **végpontjai**. A portok használata teszi lehetővé, hogy **egy adott számítógépen futó alkalmazások** **- ugyanazt a hálózati erőforrást használva - a beérkező csomagokból csak a nekik szóló csomagokat kapják meg.**

**A server és a kliens ugyanazt a portot kell használja az adott szolgáltatásra a sikeres kommunikációra.**

Egy port mindig **hozzáfűződik a házigazda IP címéhez és a kommunikáció protokoll-típusához**.

**0-s és 1024-es** számú közötti portok vannak meghatározott **szolgáltatásoknak** fenntartva.

**Melyik a 8080-as port?**

**A web böngésző kommunikációja a 80-as porton történik.** Amennyiben **a böngészőt a sajátgép szolgálja** ki(=personally hosted web server), akkor a használt port a **8080-as.**

**Mi az a jSON?**

=JavaScript Object Notation

**Kisméretű., szöveges alapú szabvány**, ember által **olvasható** **adatcserére**. **nyelvfüggetlen**; legtöbbször **server és kliens számítógép közötti adatátvitelre használják**, az xml egyik alternatívájaként. Egyszerű adatstruktúrák és tömbök reprezentálására.

**Éles alkalmazásnál hol lesz a Tomcates webkonténer?**

Éles alkalmazásnál olyan host kell (sajátgép helyett), ami folyamatosan működteti a webkonténert. Erre léteznek java hosting servicek, amivel igénybe tudunk venni egy hostot(a szolgáltató server gépét); itt is azzal kell kezdeni, hogy felkerül a hostra a Tomcat.

**A konténer, mint server elérhetővé teszi a benne lévő tartalmat a hálózaton.** http webserver, ezért [webböngészőkkel](https://hu.wikipedia.org/wiki/Webb%C3%B6ng%C3%A9sz%C5%91) http protokollon keresztül tudunk hozzá csatlakozni.

**Hogyan érhető el a felhasználó számára a webes Java alkalmazás?**

A **webserverek** és rajta keresztül az alkalmazások a **felhasználó számára böngészőn keresztül érhetőek** el (=kliens). **A felhasználó által eszközölt műveletek (adatlekérés, törlés, módosítás, új létrehozása) http requesteket generálnak**, amiket a webböngésző továbbít a server oldal (a backend alkalmazásunk)felé. A HTTP protokoll szerint a request message-ek az elvégzendő műveletek függvényében nyolcféle metódust definiálhatnak. (POST, PUT, DELETE, UPDATE)

**Mi az a protokoll?**

Az informatikában a protokoll egy **egyezmény, vagy szabvány, amely leírja, hogy a hálózat résztvevői miképp tudnak egymással kommunikálni**. Ez többnyire a kapcsolat felvételét, kommunikációt, adat továbbítást jelent: milyen sorrendben milyen protokoll-üzeneteket küldhetnek egymásnak a csomópontok, illetve az üzenetek pontos felépítését, az abban szereplő adatok jelentését is megadja.

Két eszköz között a kommunikációt általában nem egy, hanem több protokoll valósítja meg. Ezek többnyire egymásra épülnek.

**Mi az a TCP/IP protokoll?**

A protokoll felépítése a rétegződési elven alapul, minden egyes réteg egy jól definiált feladatot végez el. s a rétegek egymás között szolgálatelérési pontokon keresztül kommunikálnak. Minden réteg csak a vele szomszédos réteggel képes kommunikálni, mivel ezek egymásra épülnek.

### [*Alkalmazási réteg*](https://hu.wikipedia.org/wiki/Alkalmaz%C3%A1si_r%C3%A9teg)

Az alkalmazási réteg a felhasználó által indított [program](https://hu.wikipedia.org/wiki/Program) és a szállítási réteg között teremt kapcsolatot. Ha egy program [hálózaton](https://hu.wikipedia.org/wiki/Sz%C3%A1m%C3%ADt%C3%B3g%C3%A9p-h%C3%A1l%C3%B3zat) keresztül [adatot](https://hu.wikipedia.org/wiki/Adat_(sz%C3%A1m%C3%ADt%C3%A1stechnika)) szeretne küldeni, az alkalmazási réteg továbbküldi azt a szállítási rétegnek.

### [*Szállítási réteg*](https://hu.wikipedia.org/wiki/Sz%C3%A1ll%C3%ADt%C3%A1si_r%C3%A9teg)

Az alkalmazási rétegtől kapott adat elejére egy úgynevezett fejlécet (angolul: header) csatol, mely jelzi, hogy melyik szállítási rétegbeli protokollal (leggyakrabban [TCP](https://hu.wikipedia.org/wiki/TCP) vagy [UDP](https://hu.wikipedia.org/wiki/UDP)) küldik az adatot.

### [*Hálózati (Internet) réteg*](https://hu.wikipedia.org/wiki/H%C3%A1l%C3%B3zati_r%C3%A9teg)

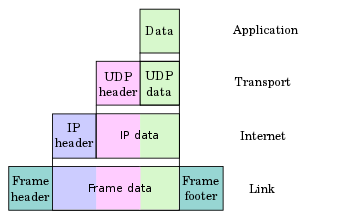
A szállítási rétegtől kapott header-adat pároshoz hozzáteszi a saját fejlécét, amely arról tartalmaz információt, hogy az adatot melyik végpont kapja majd meg.

### [*Adatkapcsolati réteg*](https://hu.wikipedia.org/wiki/Adatkapcsolati_r%C3%A9teg)

Az adatkapcsolati réteg szintén hozzárakja a kapott adathoz a saját fejlécét, és az adatot keretekre bontja. Ha a kapott adat túl nagy ahhoz, hogy egy keretbe kerüljön, feldarabolja és az utolsó keret végére egy úgynevezett tail-t kapcsol, hogy a fogadó oldalon vissza lehessen állítani az eredeti adatot.

### [*Fizikai réteg*](https://hu.wikipedia.org/wiki/Fizikai_r%C3%A9teg)

A fizikai réteg továbbítja az adatkapcsolati rétegtől kapott kereteket a hálózaton.

[](https://hu.wikipedia.org/wiki/F%C3%A1jl:UDP_encapsulation.svg)

A fogadó oldalon ugyanez a folyamat játszódik le visszafelé, míg az adat a fogadó gép alkalmazásához nem ér.

Eredetileg a fizikai és az adatkapcsolati réteg egy réteg volt, neve „Hoszt és hálózat közötti réteg”.

**Mi az a http protokoll?**

[TCP/IP](https://hu.wikipedia.org/wiki/TCP/IP) réteg felett helyezkedik el.

**Kérés-válasz alapú protokoll kliens és szerver között: a böngészők**[**HTTP**](https://hu.wikipedia.org/wiki/HTTP)**protokoll segítségével éri el a kiszolgáló webszervert; HTTP a web protokollja.**

**HTTP protokoll a request alapján nyolcféle metódust definiál. A metódusok (más szóval verbek) a megadott erőforráson végzendő műveletet határozzák meg.**

1. Step 1: Direct browser to URL. ...
2. Step 2: Browser looks up IP. ...
3. Step 3: Browser sends HTTP request. ...
4. Step 4: Host sends back HTTP response. ...
5. Step 5: The browser renders the response. ...
6. HTTP and TCP/IP.

**Mi a web konténer?**

Másk neve servlet konténer. A web szerver része; Java servlet objektumokat vezérli: menedzseli az életciklusukat és urleket rendel hozzájuk

SpringBootos alkalmazás esetén az alkalmazás része lesz, becsomagolódik a jar fileba. **Indításkor telepítésre kerül a webkonténerbe az alkalmazás és ebben fut; lepéldányosódnak a servlet objektumok a megfelelő urlek hozzárendelésével**. Az alkalmazás ha http kérést kap, a konténer előveszi a megfelelő servletet; [HttpServletRequest](https://www.studytonight.com/servlet/servlet-request.php) and .[HttpServletResponse](https://www.studytonight.com/servlet/servlet-response.php" \t "_blank) objektumokat gyárt, elvégzi velük a request-response

**Mi az a web service?**

olyan szolgáltatás, server, XML vagy jSON válaszokat képes adni egy requestre és általában kliensek vagy webalkalmazások használják; a kliens lép interakcióba a felhasználóval; web servicen keresztül egységes autentikáció a klienseknek arra, hogy van joguk az adatok megtekintéséhez és a jogokat tudja rendszerezni.

Web servicet SOAP és REST eljárással lehet elérni, hogy adatot kérhessünk

**Mi az a microservice?**

**egy adott feladatért felelős kis back service,** **sok ilyenből épül fel aztán egy nagy projekt**; el tudjuk különíteni a serviceket, akár külön servereken -> tudjuk a servert mint erőforrást optimalizálni, mikor mi mennyit kell fusson stb. (=részecskék skálázhatóak legyenek)

külön tudjuk a microserviceket tesztelni; csapaton belül elkülöníthető; minden alkosson kis egységet, ezek kommunikáljanak egymással, de legyenek elszeparálva

**Mi az a servlet?**

java objektum, ami tud http kérést feldolgozni és tud is rá választ adni; képesek vagyunk vele tartalmat generálni. A kliens böngészőjével kommunikál.

**Tomcat okosságok**

**Keretrendszer amin képes a Spring futni.** A Tomcat a vászon, a spring az ecset és én festek.

Részei:

Catalina konténer: itt fut a kódom és benne a servlet

Coyote: figyel az érkező kérésekre, olyan, mint egy kapu

*Tomcat működése*

servlet.xml fileba néztünk bele

ha a böngésző rácsatlakozik a tomcat serverre, melyik porton tudja elérni őt; default a springnél 8080

parancssorból elindítottuk a servert; a server a parancssoros terminálon keresztül logol, üzen

a server a responsait a böngészőmbe küldi, , mert azt a portot állítottuk be?

**Java alkalmazást hogy tudok egy serverre felrakni hogy eljusson a weben másokhoz is, ne csak a sajátgépen fusson?**

A felhasználó böngészőjében mutatjuk meg a programunkat. PL az Amazon egy program, a weboldala csak egy kapu.

Böngésző, vagy, mobilalkalmazás is lehet pl kapu, és mindkettő mögött uaz a program

a háttérben egy server fut, ami az egészet kiszolgálja, fogadás, küldés, kalkulálás, tárolás.

a kiszolgáló, server, mint teljes intézmény. Eléréséhez Ismernünk kell a címét(ip), de ez nem egyértelmű, ezért vannak a domain nevek.

adatcsomagok a neten; a holnap nem jön át egybe a felhasználó gépére

kommunikáció kétféleképpen:

UDP, csak áramlik az adat, nincs visszajelzés, kimaradhat ez az (sport közvetítés)

TCP – nem maradhat ki adat, lassabb a kommunikáció, folyamatos visszakérdezés, hogy megérkezett-e?

fix ip című tartalom (weboldalak), dinamikus ip – routerek

ha server megkapja a lekérdezést, válaszát feldarabolva visszaküldi az adott ipre; a routeremre megérkezik és ő küldik tovább valamelyik gépre

mind a response, mint a request két részre bontható: **header: fejléc; body:** magát az üzit tartalmazza

a böngésző szerepe a folyamatban, hogy megkapja a html kódot és abból összeállítja az oldalt; a böngészőn áll, hogyan értelmezi az üzenetet

a böngésző is végez munkát és a server is; egyre jobb sajátgépek mellett egyre jobban terhelhetőek a böngészők is, egyre több kódot futtatnak -< itt jön épbe a javaScript, ami a böngészőn belül képes futni és független a java servertől.

nem kell elolvasni az üzenetek bodyját ahhoz, hogy tudjuk, gebasz, van, elég hozzá a **header is. A header tartalmaz egy státusz kód részt**, amiben látjuk, hogy milyen jellegű választ kaptunk a szervertől. HTML státuszkódok:

1xxx – informatív jelzés

2xx – a kérés elfogadva

3xxx – átirányítás

4xx – kliens hibák

5xx – server oldali hiba



**cashelés** – nem tölti be minden egyes alkalommal, hanem el van tárolva és megjeleníteni (=cashelve van); a cashelés mellett a tartalom viszont nem frissül; van itt ezért nem érdemes

web server nem biztos hogy csak htmlt, csst és javascriptet ad vissza – van olyan server, ami egyáltalán nem ad vissza htmlt

böngészőnek érdemes tudnia, mit tlt le, mert pl meg is tudja akkor nyitni; ezért a headerben van a **Mime type**, ami megmondja, hogy mi a tartalom(html, szöveg, videó, mp3., stb)

native alkalmazás – nem htmlt kap, hanem xml és jSon kevés tartalommal, mert majd a saját eszközeivel fogja az alkalmazás a grafikai elemeket hozzáadni, ilyenek pl a mobilalkalmazások

**Milyen formában jelenik meg az adat az informatikában**

**Servlet és Tomcat**

**web alkalmazás**

a felhasználót szolgálja ki

**DispatcherServlet**

fő, irányító szervlet

The main purpose of the dispatch() method is to find an appropriate handler for the request;  Spring goes through the registered implementations; he most widely-used class for handler mapping is RequestMappingHandlerMapping, which maps a request to a @RequestMapping-annotated method of a @Controller class.

User sends request for a servlet by clicking a link that has URL to a servlet

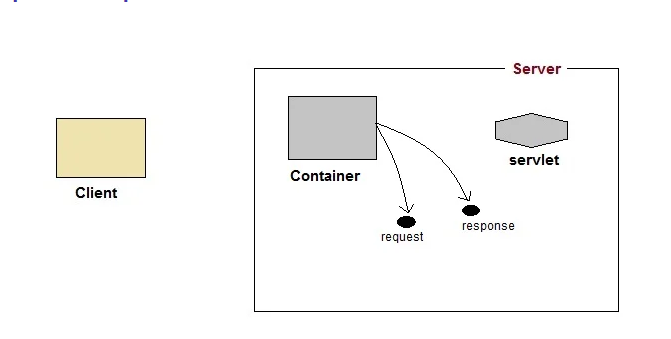
The container finds the servlet using deployment descriptor and creates two objects : [HttpServletRequest](https://www.studytonight.com/servlet/servlet-request.php) and .[HttpServletResponse](https://www.studytonight.com/servlet/servlet-response.php" \t "_blank)

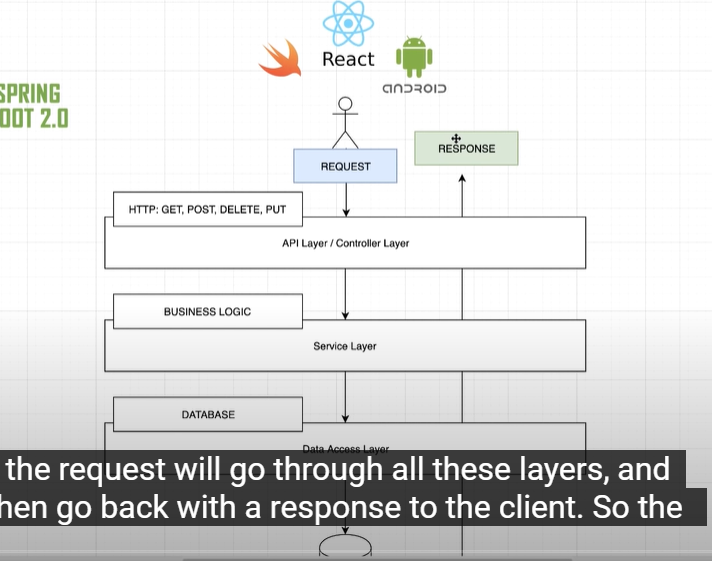
Then the container creates or allocates a thread for that request and calls the Servlet's service() method and passes the **request, response** objects as arguments.

The service() method, then decides which servlet method, doGet() or doPost() to call, based on **HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the service() will call Servlet's doGet() method.

The service() method, then decides which servlet method, doGet() or doPost() to call, based on **HTTP Request Method**(Get, Post etc) sent by the client. Suppose the client sent an HTTP GET request, so the service() will call Servlet's doGet() method.

After the service() method is completed the **thread** dies. And the request and response objects are ready for **garbage collection**.

****

****

**aplicaiton properties**

itt lehet különböző környezetekre konfigurációt megadni ()test, production environment

**Mi az az API?**

(=Applicaiton Programming Interface; alkalmazásprogramozási interfész) **program vagy eszközkészlet, amit más programok használhatnak anélkül, hogy annak belső működését ismernie kellene**. Általában nem kötődik [programozási nyelvhez](https://hu.wikipedia.org/wiki/Programoz%C3%A1si_nyelv): bármilyen programnyelvből lehetséges az API meghívása, amennyiben a megfelelő [paramétereket](https://hu.wikipedia.org/w/index.php?title=Param%C3%A9ter&action=edit&redlink=1) a hívás biztosítja. Az API eredetileg az ilyen programok/eszközkészletek dokumentációját jelenti.

**Mi az a Fluent API?**

Amikor az API által generált eredmény metódushívások láncolásával történik.

**Kérdések**

Hogy történik gyakorlatba, amikor az alkalmazás már élesbe fut; hol van ilyenkor a web konténer(eddig a sajátgépen volt),; egy hoston de az hol szokott lenni, java hosting? -> Tomcat hosting platforms: A2, Kamatera; van-e ingenes?

Frontend bekapcsolásával hova küldi a backend a responset a http response body helyett?

Browser refresh egyenlő a GET-el?

Web serverhez csak böngészővel lehet csatlakozni?

Az alkalmazás frontend tartalma( html, css, script) és a backend tartalom külön serveren szokott lenni? a Böngésző éles alkalmazásnál a frontendes tartalommal kommunikál, az pedig a backenddel?