**Validáció**

**(def, BV 2.0 JSR 380 szabvány; magán az adaton legyen megadva(entitás vagy DTO); megadás anotációval; beépített és saját**

SpringBoot támogatja a **bemenő adatok ellenőrzését = validáció**

P. egy employee létrehozásakor validálni kell, hogy a neve nem üres String, kötelező legyen megadni

A **SpringBoot** beépítetten tartalmazza a **Bean Validaiton 2.0 támogatást** és annak az implementációt

Bean Validation: különálló **szabvány** (száma: JSR 380) , ami a Java EE része is; **különböző java objektumokhoz történő ellenőrzéseket, validációkat lehet vele definiálni**

A BV lényege, hogy a **validáció ne réteghez legyen kötve/implementálva**, hanem **magán az adaton legyen megadva**: egy Springes alkalmazás esetén azokon a **Dto-kon**, amelyek a **Controller metódus paramétereként** szerepelnek

Validációt megadni **annotációval** lehet, az adott **Dto** **osztályra** vagy a Dto mezőire(=**attribútumaira**)

vannak **beépített** és **saját annotációk** is

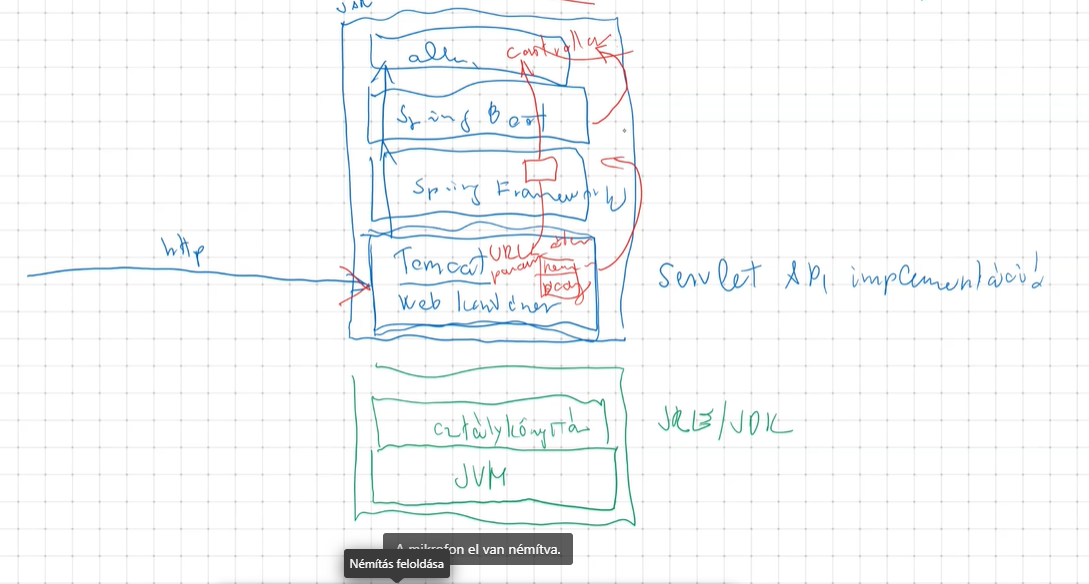
meg lehet adni, hogy a **validáció mikor fusson**; pl. ha egy entitásra tesszük, akkor az adatbázisba való lementés előtt, ha egy Dto-ra tesszük, akkor a Controller metódus meghívásakor: **a validáció meghatározása a Dto-n vagy az entitáson történik**, és később lehet megmondani, hogy ezek **hol kerüljenek meghívásra**

**Servlet!**

**jar állomány:**

**alul Tomcat – bejövő http kérépseket fogadja, felette Spring framework; a felett egy SpringBoot és azon felül az alkalmazás; tomcat a http kérést átadja a springnek , ő meg az alkalmazásnak**

**Tomcatben van a servlet; a Tomcat egy Servlet API implementáció (=web konténer); a Spring imlementál servleteket amik a Tomcaten futnak és ezek a servletek hívják meg a controllert; a http kérés szöveg headerrel meg bodyval; Tomcat olvassa ki az urlből a paramétereket, adja át springnek , spring kicsomagolja jsonből java objektumot csinál és ezzel hívja meg a controllert**

****

**Hogy épül föl egy alkalmazás?**

Jar állományt elindítjuk, azon belül Tomcat, Spring különböző részei és maga az alkalmazás, ami több rétegű (Controller, Servic,e Rep)

http protokol:: kliens server felépítésű; kliens = böngésző, ő hívja meg az alkalmazást ő a server

a http kérés-válasz alapú: request és response; ezek külön messagek; az egész egy tcp/ip protokoll fölött zajlik; a http ennek a szolgáltatásait igénybe veszik

mindig a kliens kezdeményezi a kapcsolatot

http üzenet részei: header és body

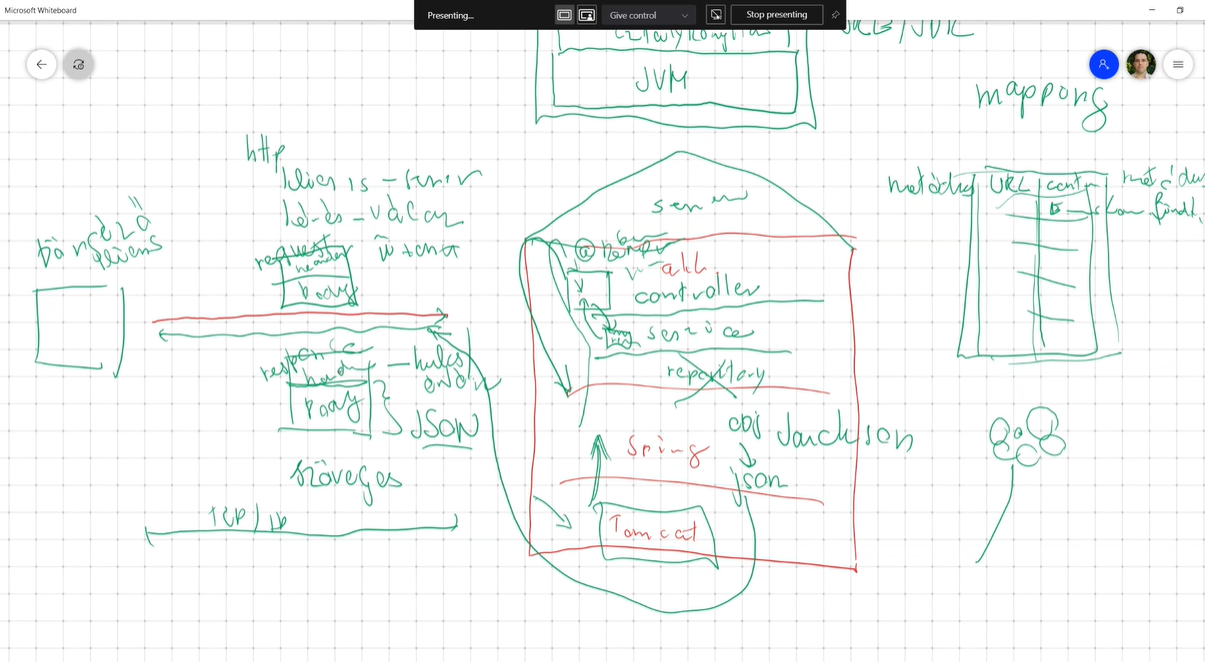
header: kulcs-érték párok; body: törzs, REST esetén json dokuk

a http protokoll üzik szövegesek(header és body)->amit mi láttunk tényleg az; egy soremelés van a header és body között, így van különválasztva

ha szöveges a protokoll, hogy töltök fel képet? képet le lehet kódolni szöveges tartalomra; de ezen a szöveges tartalmon amúgy bináris adatot is át lehet nyomni

a Spring hogy találja meg, melyik controller kell? @RequestMapping alapján

háttérben map arra, hogy milyen urlhez és http metódushoz melyik controller melyik metódusa tartozik; A Spring ezen beépített maping táblája alapján fogja tudni melyik metódust kell meghívnia



**Beépített annotációk**

****

**Validáció Controlleren**

**(cél: metódus hívásra automatikus validáció az objektumon(még egyenlőre Dto-kon;metódus paramétr elé @Valid, DTO osztályban atrira beépített vagy saját annotáció; jsONt ad vissza sok infóval; ez nem szabványos, át kell konvertálni Problembe csomglva: ehhez tundi kell: MEthodArgumentNotValidExceptiont dob a sprong validációs hiba esetén;ezt kell elkapni az exceptionHandler metódusban; majd getBindigResult().getFieldErrors().stream alapján Violation objektumokat gyártunk le, és ezt mint új mezőt felvesszük a problem objektumban**

*Hogyan érjük el azt, hogy a Spring automatikusan vizsgálja meg a Dto paramétereket, amikor az adott metódus meghívásra kerül?*

Az adott **paraméter elé** el kell helyezni a **@Valid** annotációt a **metódusban**, **a Dto osztályban** pedig a name **atri elé** @NotNull annotáció, amiben a hibaüzt is meghatározzuk



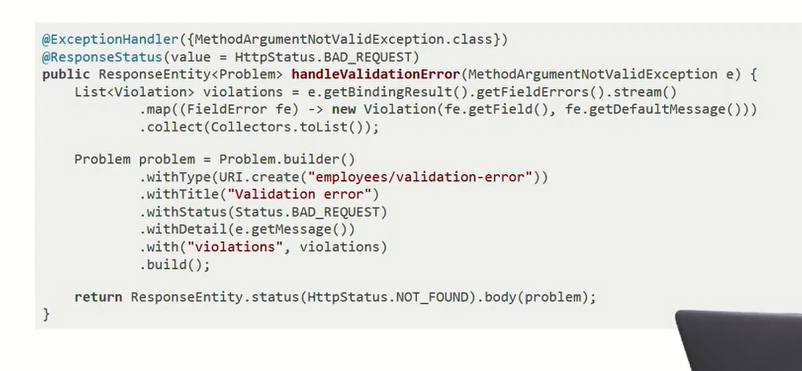
ami **visszamegy**, az egy **jSON**, ami részletesen tartalmazza a **validáció eredményét**, ami nagyon **sok infót tartalmaz**(pl. mikor történt, státuszkód(400), mezőként a hibák; ez a **részletesség azért jó,** mert a felületen mezőként tudjuk majd megmondani, mi volt a hiba; pl. a névvel, címmel, telszámmal stb.



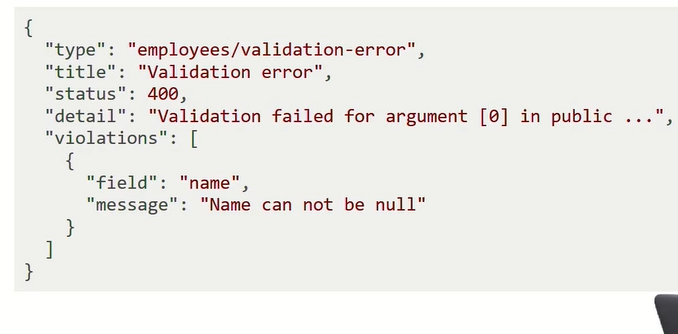
de ez a **Spring saját ábrázolása és nem szabványos**; ha **szabványosat szeretnénk** visszaküldeni, akkor itt is a **Problem third party libraryt** használhatjuk, egy **exceptionHandlert** kell deklarálni

Bonyinak tűnik elsőre; a **Springnek tudni kell a belső működését validációs hiba esetén**, ennek menete:

megpróbálja meghívni a metódust, ami van rajta paraméter, azt megpróbálja validálni, ha ez nem sikerül, dob egy **MEthodArgumentNotValidExceptiont; a handler metódussal mi ezt kezeljük le;** a mezőkre vonatkozó hibaüziket a **getBindigResult().getFieldErrors()-al** lehet lekérni; ezeket **egy Violation listába(String field, String message) átkonvertáljuk**; végül pedig ezt, **mint egy egyedi mező, felvesszük a problem objektumon belül**

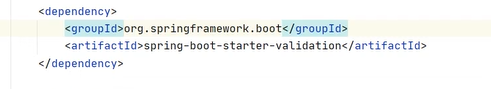


Amikor ezt a problemet visszaadjuk, a Spring ezt jSONre szerializálja, és így fog kinézni:



**Validáció – gyakorlat**

1. Bean Validaiton felvétele függőségként a pom.xmlbe:



1. a createEmployeeCommandon belül a name attribútumon a Bean Validaiton egy beépített annotációját a @NotBlanket használjuk, messagge-el
2. meg kell adni, hogy a validáció mikor menjen végbe? controllerben a createEmployee() metódusban a @RequestBody elé még kell a @Valid is
3. alkalmazás elindítása és beküldünk egy érvénytelen POST kérést -> http fileban új request hozzáadása és name: ””; a request küldésekor a springBoot egy részletes jSONt ad vissza(field és rejectedvalue); ez a Spring saját formátuma; de mi az RFCs szabványnak megfelelő hibaüzit akarunk, problem-et használva->
4. exceptionhandler implementálása, ami elkapja a MethodArgumentNotValidException t, amit a Spring ilyenkor dob

kerüljön visszaadásra a mezőnek a neve és a hibaüzi: új Violation osztály létrehozásafields és message atrikkal lombokkal, majd az exceptionból lekérdezzük a prolémákat (getBindigResult().getFieldErrors().stream().map->new Violationök listába, majd ezt becsomagoljuk egy problem példányba (.builder, .witthype, .withStatus, .withdetail, **.with(„violations”, violaitons),** .build -> végül ResponseEntityvel térünk vissza: .status, .contenttype, **.body(problem)**

1. http request beküldése újra: ezzel nem a Spring beépített jSON formátumát hanemaz RFC szabvány szerinti megfelelő jSONt

**Validáció – gyakorlat – saját validáció**

A BV lehetővé teszi saját validációk írását; ehhez kell egy annotáció és egy osztály, amiben implementáljuk a validáció módját

1. *@Name annotáció létrehozása*

@kerüljön rá a @Retention(RetentionPolicy.RUNTIME)=futás közben feldolgozható legyen; és a @Target(ElementType.FIELD)=mire tehető rá

1. *attribútumok*

String message() default ”Invalid name”; -> ez később akár felülbírálható ott, ahol használom az annotációt

Class<?>[] groups default {};

Class<? extends PayLoad>[] payload() default {};

int maxLength() default 50;

@Constraint annotáció(validateBy = NameValidator.class) még rája; = validátorhoz tartozó annotáció

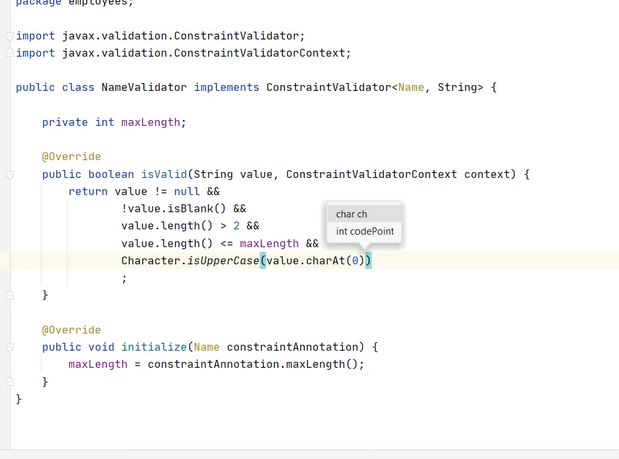
1. NameValidator osztály létrehozása extends ConstraintValidator<Name, String>

egy atri: private int maxLength;

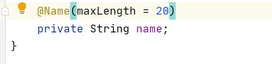
1. isValid() és initialize() metódusok implementálása

isValid() ne legyen érvényes nullra és egyéb validációk

1. használatba vétel: CreateEmployeeCommandban a nameről levesszük a @NotBlanket, és rátesszük a @Name-t



1. @Name további paraméterezése, alapértelmezett értékek felülbírálásával:

****

ugyanez a messagere

1. http requestek beküldése és csekk

**Kérdések**

*Mire használjuk a validációt?*

*A SpringBoot hogyan támogatja a validációt?*

*Mi az a Bean Validaiton és mi a lényege?*

*Hogy adunk meg validációt és hol kell elhelyezni?*

*Milyen típusai vannak a validációs annotációknak?*

*Milyen típusú beépített validációs annotációkat ismersz?*

*Hogy adjuk meg, hogy mikor fusson a validációs ellenőrzés és jellemzően mikor fut?*

*Hogy szabjuk személyre a validációs hibaüzenetet?*

*Mit ad vissza a validáció sikertelen vlaidáció esetén? Mi a státuszkód?*

*Hogyan küldünk vissza szabványos validációs üzenetet?*

*Milyen exceptiont dob a Spring, ha a paraméter tnem sikerül validálni?*

*Az előző kettő vonatkozásában hogy írunk meg egy exceptionHandlert? Mire jó a Violation osztály? Mi lesz a handler visszatérési értéke?*

*A visszatérési érték miben lesz lesz szerializálva?*