

Készítette: Dr. Subecz Zoltán

Neumann János Egyetem, GAMF Informatikai és Műszaki Kar, Informatika tanszék

subecz.zoltan@gamf.uni-neumann.hu

Java alkalmazások - Alapképzés - gyakorlatok

Tartalomjegyzék

Bevezetés - Csak olvasmány	3
Első feladat - konzol alkalmazás készítése	3
Bevezető feladatok - Ismerkedés a Java-val, IntelliJ-vel - változók, ciklusok, kiíratás, beolvasás – Csak olvasmány	5
1. gyakorlat - Metódusok, fájlkezelés, összetett adatszerkezetek – Array, List, Set, Map.....	12
1-4 egyszerű feladatok – Csak olvasmány - Fájlból olvasás Array-be, Set-be, List-be, Map-ba -	
Lényege előadáson volt	12
Összetett feladatok - Array, Set, List, Map, Class	16
2. gyakorlat – Objektum orientált programozás	24
3. gyakorlat – JDBC – Adatbáziskezelés.....	31
Java Spring, Spring Boot, Bevezetés - Csak olvasmány	39
4. gyakorlat - Egyszerű webalkalmazások - Spring.....	41
Feladat-1 - @SpringBootApplication, @Controller, Route, @GetMapping, @ResponseBody, JSON .	41
Feladat-2 - olvasás HTML Template-ből	48
Feladat-3 - HTML template módosítása futás közben - Model, Thymeleaf – MVC	49
Spring alkalmazás futtatása IntelliJ nélkül – JAR fájl készítésével – egyelőre csak olvasmány	51
5. gyakorlat - 1. ZH a Spring előtti anyagból	51
6. gyakorlat - Spring Űrlapok	52
Feladat-1 - Űrlap, Model, @GetMapping, @PostMapping,	52
Feladat-2 - Űrlap adatainak szerver oldali validációja	58
7-8. gyakorlat.....	61
A: Spring-Boot-Adatbázis-JPA - Hibernate-MySQL-CRUD alkalmazás	61
B: Többtáblás feladat.....	72
1, Táblák közötti kapcsolat nélkül (Fooldal részhez)	75
2, Táblák közötti kapcsolattal	79
9-10. gyakorlat - Spring-Boot – Security – Roles, JPA MySQL - ZH-BAN NEM LESZ, CSAK A	
BEADANDÓHOZ KELL	82
1. feladat	86
User, Role, Message JPA Entity-k	86
Repository-k	87
A Security beállításai – Authentication, Authorization	88
Controller.....	89
Nézetek Thymeleaf -el.....	90

Kiegészítő feladat - BCrypt kód generálás	91
2. feladat: Kiegészítés regisztrációs oldallal.....	92
11. gyakorlat - Spring-Boot – RESTful API - JPA MySQL	96
Bevezetés - Csak olvasmány - Kihagyható	96
REST, RESTful	96
REST, RESTful, cURL, fake REST API, Tesztelés cURL-el, Postman-el.....	98
A feladat.....	105
Az alkalmazás tesztelése cURL -el.....	110
Az alkalmazás tesztelése Postman -el.....	111
Gyakorló házi feladat – Kiegészítés kliens oldali alkalmazással	113
12. gyakorlat: 2.ZH a Spring anyagából.....	114

Bevezetés - Csak olvasmány

- **A tantárgy tantervi helye: 5. félév**
- Már több programozási nyelvet tanultak eddig
- **A Java alapok nagyon hasonló a C# alapokra, amit már tanultak.** Itt azokat már nem tanuljuk újra: a különbségeket emeljük ki és példákat nézünk, amik segítik a gyakorlatok és a további fejezetek példáinak megértését.
- Nem csak Java alapokat tanulunk:
a tantárgy neve: **Java alkalmazások**
-

Ajánlott irodalom:

- Nagy Gusztáv: <http://nagygusztav.hu/java-programozas>
- Nagy Gusztáv: <http://java.progtanulo.hu/>
- Angster Erzsébet: Objektumorientált tervezés és programozás, JAVA, 1-2. kötet
- Spell, B.: Pro Java 8 Programming, Apress, ThirdEdition, California, 2015.

Otthoni gépre telepítsék fel a **JDK-t (Java Development Kit)**

<https://www.oracle.com/java/technologies/downloads/>

A feladatokat az **IntelliJ IDEA**-ben fogom bemutatni, de használhatnak más fejlesztőkörnyezetet az órán és a ZH-nál is pl. Netbeans, Eclipse, MS Code, NotePad++, Komodo edit, Jegyzettömb, ...

lásd: **IDEA letöltése és telepítése oktatási célra.pptx** fájl.

IntelliJ IDEA

Betűméret állítás: File menü / Settings / Editor / General / Font

Első feladat - konzol alkalmazás készítése

Készítsük el az egyszerű **Hello world!** alkalmazásunkat.

- Mappa létrehozása a project-nek Pl. **c:\Java-1**
- IntelliJ IDEA indítása
- New project / Java /
- Project name: Elso
- Project location: C:\Java-1
- Nézzük meg a létrehozott mappákat, fájlokat. src mappa még üres
- **Új osztály hozzáadása:**
src mappán állva: File menü / **New / Java class:** HelloWorldApp
- Írjuk be: main

```
public class HelloWorldApp {  
    main  
}
```
- Válasszuk ki az IntelliJ által felajánlott **main() method declaration** kód-kiegészítési lehetőséget
- IntelliJ kiegészíti:

```
public class HelloWorldApp {  
    public static void main(String[] args) {  
  
    }
```

- ```

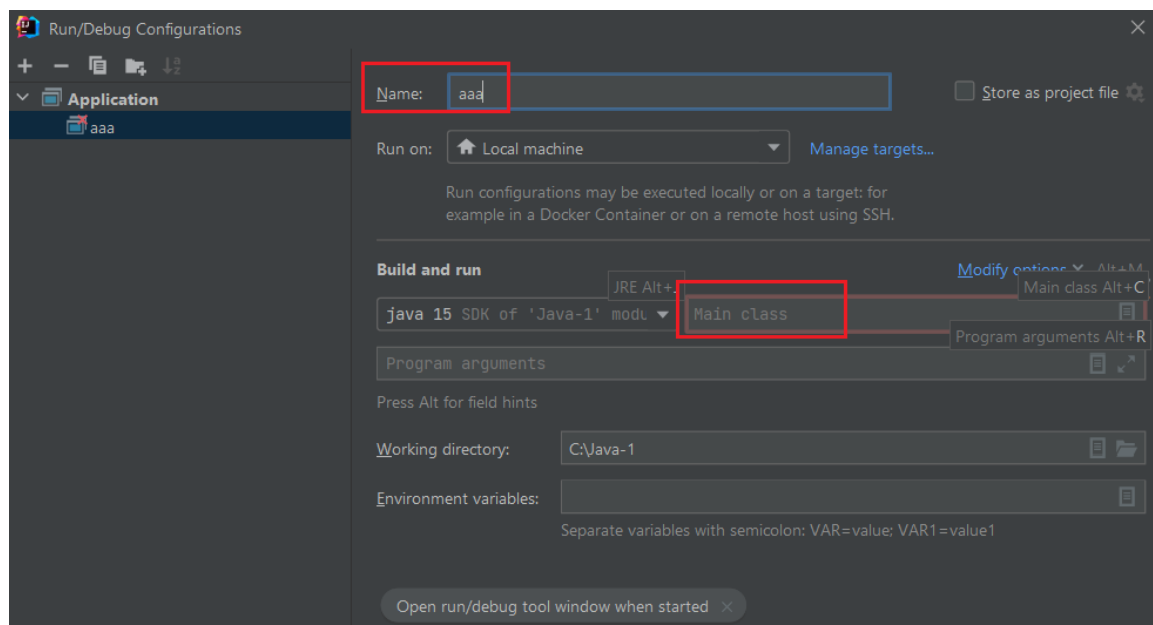
}

```
- Töltsük ki a metódust:  

```

public class HelloWorldApp {
 public static void main(String[] args) {
 System.out.println("Hello world!");
 }
}

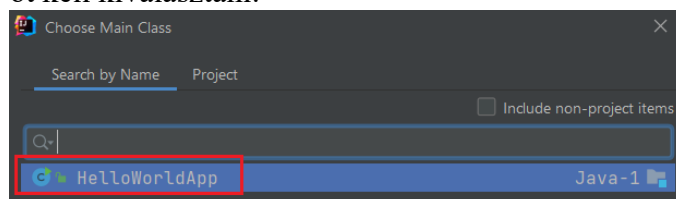
```
  - System.out.println(); gyorsabban:  
Írjuk be sout + Enter => Beírja: System.out.println();
  - **Ha a futtatási zöld gomb nem aktív:**  
Run menü / Edit configurations / Add new run configuration / Application

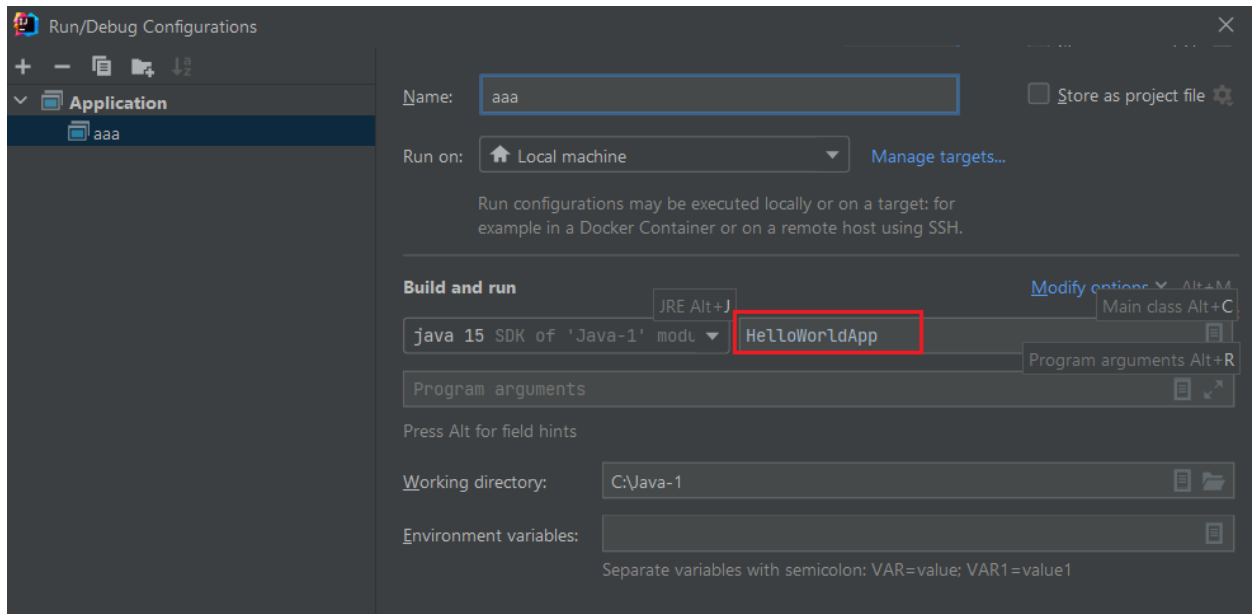


A Main class-nál meg kell adni, hogy melyik osztály tartalmazza a main() metódust.

Belépve ebbe a választásba:

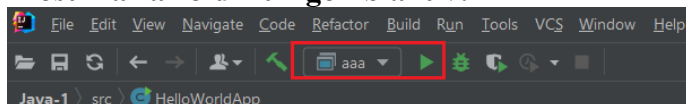
Várni kell, amíg a program megkeresi az osztályokat a project-ben. A felajánlott HelloWorldApp-ot kell kiválasztani:





OK

**Most már a zöld Run gomb aktív:**



Futtassuk az alkalmazást

Kiírja: Hello world!

Az előző módszert kell akkor is alkalmazni, ha több osztály is tartalmaz main() metódust: meg kell adni, hogy indításkor melyiket futtassa.

**Keressük meg a létrehozott class fájlt.**

out\production\Java-1\HelloWorldApp.class

- Futtatás parancssorból:**

c:\Java-1\out\production\Java-1>java HelloWorldApp

Kiírja: Hello world!

## Bevezető feladatok - Ismerkedés a Java-val, IntelliJ-vel - változók, ciklusok, kiíratás, beolvasás – Csak olvasmány

Néhány egyszerű feladat a Java szintaktikájának a megismeréséhez.

### 1. feladat

Hozzon létre 4 változót, ebben a sorrendben: egész, hosszú egész, valós és byte! A változóknak adjon tetszőleges kezdőértéket is! Ugyanilyen sorrendben írja is ki a változók értékét!

Keressük meg az elkészített .java és .class fájlokat.

### Megoldás

#### Feladat.java

```
package csomag1;
```

```
public class Feladat {
 public static void main(String[] args) {
 int a = 3;
 long b = 34573799;
 double c = 3.67;
 byte d = 110;
 System.out.println("a=" + a + " b=" + b + " c=" + c + " d=" + d);
 }
}
```

A kiíratást így is írhatjuk:

```
System.out.println(String.format("a=%d b=%d c=%.2f d=%d",a,b,c,d));
```

vagy a paraméterek sorszámozásával:

```
System.out.println(String.format("a=%1d b=%2d c=%3.2f d=%4d",a,b,c,d));
```

## 2 feladat

Hozzon létre egy-egy változót a

- neve
  - életkora
  - szemüvegessége (igen vagy nem)
  - magassága
- tárolására

Használjon beszédes változóneveket! Adjon értékeket a változóknak! Amelyik változónál van értelme, állítsa véglegesre. Írassa ki a változók értékeit egy kis magyarázattal együtt!

## Megoldás

```
package csomag1;
```

```
public class Feladat {
 public static void main(String[] args) {
 final String nev = "Kovács Ferenc";
 int eletkor = 35;
 boolean szemuveges = true;
 double magassag = 178.5;
 System.out.println("nev=" + nev + " életkor=" + eletkor + " szemüveg=" + szemuveges +
" magasság=" + magassag);
 }
}
```

## 3 feladat

Hozzon létre két egész változót tetszőleges értékkel. Hozzon létre egy logikai változót, amely igaz, ha két egész egyenlő, és hamis különben. Írassa ki a logikai változó értékét.

## Megoldás

```
package csomag1;
```

```
public class Feladat {
```

```

 public static void main(String[] args) {
 int a=5, b=3;
 boolean egyenlo = a==b;
 System.out.println("Egyenlő-e a két változó?: " + egyenlo);
 }
}

```

#### 4 feladat

Olvasson be a billentyűzetről egy pozitív egész értéket. A program állapítsa meg a szám jegyeinek számát!

#### Megoldás

```
package csomag1;
```

```
import java.util.Scanner;
```

// Ezt nem kell magunknak beírni, a Netbeans beírja, ha a piros aláhúzásnál rákattintunk, majd Alt-Enter

```

public class Feladat {
 public static void main(String[] args) {
 Scanner olv = new Scanner(System.in);
 System.out.println("Adjon meg egy egész számot:");
 int szam = olv.nextInt(); // Ez is jó: Integer szam = olvas.nextInt();
 System.out.println("A számjegyek száma: " + Integer.toString(szam).length());
 }
}

```

Scanner: jó billentyűzetről és fájlból való beolvasásra is.

int: primitív adattípus

Integer: osztály (class)

#### 5 feladat

Olvasson be két egész számot. Számítsa ki két szám legnagyobb közös osztóját.

Például: lko(12, 18) = 6, lko(10, 5) = 5

#### Megoldás

a, megoldás, for ciklussal:

```
package csomag1;
```

```
import java.util.Scanner;
```

```

public class Feladat {
 public static void main(String[] args) {
 Scanner olv = new Scanner(System.in);
 System.out.println("Adjon meg egy egész számot:");
 int szám1 = olv.nextInt();
 System.out.println("Adjon meg még egy egészet:");
 int szám2 = olv.nextInt();

 int lko;

```

```

 for(lnko=Math.min(szám1, szám2);lnko>0;lnko--)
 if(szám1%lnko == 0 && szám2%lnko == 0)
 break;
 System.out.println("Legnagyobb közös osztójuk="+lnko);
 }
}

```

**b, megoldás:** Euklideszi algoritmussal: hatékonyabb megoldás

A két szám közül nézzük meg, melyik a nagyobb, és abból vonjuk ki a kisebbet. Ezt addig ismételjük, amíg a két szám egyenlő nem lesz. Ekkor megkapjuk a keresett lnko-t.

```

while(szám1 != szám2)
 if(szám1>szám2) szám1 -= szám2;
 else szám2 -= szám1;
System.out.println("Legnagyobb közös osztójuk="+szám1);

```

## 6 feladat

Olvasson be három valós számot. Döntse el a három számról, hogy mennyi a legnagyobb érték. Csak 3 változó és elágazások használhatók!

### Megoldás

```

package csomag1;

import java.util.Scanner;

public class Feladat {
 public static void main(String[] args) {
 Scanner olv = new Scanner(System.in);
 System.out.println("Adjon meg három számot:");
 System.out.println("a=");
 double a = olv.nextDouble();
 System.out.println("b=");
 double b = olv.nextDouble();
 System.out.println("c=");
 double c = olv.nextDouble();

 System.out.print("Legnagyobb=");
 if(a>=b && a>=c)
 System.out.println(a);
 else {
 if (b>= c && b>= a)
 System.out.println(b);
 else
 System.out.println(c);
 }
 }
}

```

## 7 feladat

Olvasson be egy egész számot. Írjon programot, mely megmondja a számról, hogy prím-e.



## Megoldás

### a. megoldás:

```
package csomag1;

import java.util.Scanner;

public class Feladat {
 public static void main(String[] args) {
 Scanner olv = new Scanner(System.in);
 System.out.println("Adjon meg egy számot:");
 System.out.println("a=");
 int a = olv.nextInt();

 boolean prim=true;
 // Elég a szám gyökéig elmenni
 for(int b=2; b<= (int) Math.sqrt(a); b++) // típuskényszerítés itt: tizedes részt levágja
 if(a%b==0){
 prim=false;
 break;
 }
 if(prim)
 System.out.println("Prím");
 else
 System.out.println("Nem prím");
 }
}
```

### b. megoldás:

A logikai változó nélkül. Kicsit más módszerrel. Itt is a szám gyökéig megy el.

```
package csomag1;

import java.util.Scanner;

int b = 2;
for (; b * b<= a; b++)
 if (a % b == 0) {
 System.out.println("Nem prím");
 break;
 }
if (b * b>a)
 System.out.println("Prím");
```

## **8 feladat**

Írja ki egy szöveges változó összes 3 betűs szeletét a substring metódus segítségével! Pl. "körte" esetén:  
kör örtte

## **Megoldás**

```
package csomag1;

public class Feladat {
 public static void main(String[] args) {
 String str="körte";
 for(int i=0; i<str.length()-2; i++)
 System.out.println(str.substring(i, i+3));
 }
}
```

### **9 feladat**

A lehető legkevesebb objektumpéldány létrehozásával állítson elő egy olyan String objektumot, amely az angol abc betűit tartalmazza. (Ciklusban növeljen egy változót a következő karakterrel.)

## **Megoldás**

A String a Java nyelvben megváltozhatatlan. Ha már egyszer értéket kapott, akkor nem lehet megváltoztatni. Ha meg akarom változtatni, akkor egy új string változót hoz létre a memóriában. Ezért helyette a StringBuilder osztályt használjuk ilyen feladatra.

```
package csomag1;

public class Feladat {
 public static void main(String[] args) {
 StringBuilder sb = new StringBuilder();
 for (char c = 'a'; c<= 'z'; c++)
 sb.append(c);
 for (char c = 'A'; c<= 'Z'; c++)
 sb.append(c);
 String str = sb.toString();
 System.out.println(str);
 }
}
```

### **10 feladat**

Olvasson be a billentyűzetről egy szöveget (String). Számolja meg a szöveges változó szavainak számát. A szavakat a szóközők választják el egymástól.

## **Megoldás**

```
package csomag1;

import java.util.Scanner;

public class Feladat {
 public static void main(String[] args) {
 System.out.println("Írjon be egy szöveget!");
 Scanner olv = new Scanner(System.in);
```

```

 String str = olv.nextLine();
 String[] strTomb = str.split(" ");
 System.out.println("A szöveg szavainak a száma="+strTomb.length);
 }
}

```

Az utolsó három sor egy sorban megvalósítva:

```
System.out.println("A szöveg szavainak a száma="+olv.nextLine().split(" ").length);
```

-----

**Új csomag hozzáadása:** File menü / New / Package

**Gyakorló feladat:**

Adjuk hozzá a következő két csomagot:

első.masodik

harmadik

Hozzuk létre a következő osztályokat:

első.masodik: Proba2

harmadik: Proba3

Az src mappában nézzük meg a létrehozott mappákat és fájlokat.

**Refactor**

Írjuk át a fő osztály nevét Foprogram-ra

Osztály nevére kattintva / Refactor menü / Rename / Szöveg átírása + Enter

Az src mappában nézzük meg a változást

**Kód generálás**

Vegyünk fel két globális változót (class utáni sorba). pl.

```
int a;
```

```
int b;
```

**a, Konstruktor generálás**

Az int b; UTÁNI üres sorba kattintsunk / Code menü / Generate / Constructor ...

**b, getter és setter metódusok generálása**

Code menü / Generate / Getter and Setter

**c, toString metódus generálás**

Code menü / Generate / toString()

# 1. gyakorlat - Metódusok, fájlkezelés, összetett adatszerkezetek – Array, List, Set, Map

**Map:** C#-ban Dictionary

További néhány egyszerű feladat a Java szintaktikájának a megismeréséhez.

Metódusok, fájlkezelés, Adatszerkezetek: tömb, lista, halmaz, map

A feladatoknál használjon metódusokat.

## Fájl és szövegfeldolgozás feladatok

1-4 egyszerű feladatok – Csak olvasmány - Fájlból olvasás Array-be, Set-be, List-be, Map-ba - Lényege előadáson volt

### 1 feladat – Fájlból olvasás, Array

Egy számok.txt szövegfájlban (**Források.zip**) van 10 egész szám egymás alatt.

A forrásfájlt másolja abba a mappába, ahol az src és a bin mappa is van.

Olvassa be a számokat egy tömbbe. Írja vissza a számokat a képernyőre és egy másik fájlba is egymás mellé szóközzel elválasztva. Számítsa ki az átlagot és írassa ki a képernyőre.

### Megoldás

a, megoldás

### Feladat.java

```
package csomag1;
```

```
// Az import-okat nem nekünk kell beírni, az IntelliJ beírja
```

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;
```

```
public class Feladat {
 static void Megold() throws FileNotFoundException{
 // throws FileNotFoundException: Ezt nem kell magunknak beírni, Netbeans beírja
 Scanner olvas = new Scanner(new File("számok.txt"));
 int[] tomb = new int[10];
 int i=0;
 while(olvas.hasNext())
 tomb[i++] = Integer.parseInt(olvas.nextLine());
 int összeg=0;
 PrintWriter kiir = new PrintWriter("számok2.txt");
 for(i=0;i<10;i++){
 System.out.print(tomb[i]+" ");
 kiir.print(tomb[i]+" ");
 összeg += tomb[i]; //összeg = összeg+tomb[i];
 }
 kiir.close();
 }
}
```

```

 System.out.println("\nA számok átlaga="+összeg/10.0);
 }

 public static void main(String[] args) throws FileNotFoundException {
 Megold();
 }
}

```

static: az osztály saját változója, metódusa. Az osztály példányosítása nélkül is elérhető. Minden osztály példánynál megegyezik. Érdekes már kezdő osztályt (itt: Feladat) példányosítani. Ilyenkor csak a main(...) metódusnál kell a static:

### **b, megoldás az osztály példányosításával.**

```

package csomag1;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;

public class Feladat {

 Feladat() throws FileNotFoundException{
 Megold();
 }

 void Megold() throws FileNotFoundException{
 Scanner olvas = new Scanner(new File("számok.txt"));
 int[] tomb = new int[10];
 String sor;
 int i=0;
 while(olvas.hasNext())
 tomb[i++] = Integer.parseInt(olvas.nextLine());
 int összeg=0;
 PrintWriter kiir = new PrintWriter("számok2.txt");
 for(i=0;i<10;i++){
 System.out.print(tomb[i]+" ");
 kiir.print(tomb[i]+" ");
 összeg += tomb[i]; //összeg = összeg+tomb[i];
 }
 kiir.close();
 System.out.println("\nA számok átlaga="+összeg/10.0);
 }

 public static void main(String[] args) throws FileNotFoundException {
 Feladat feladat = new Feladat();
 }
}

```

Itt a Feladat() metódus a Feladat osztály konstruktora.

## **2. feladat – Fájlból olvasás, Set**

Az adatok.dat fájlban egészek vannak egymás alatt. Lehetnek köztük azonosak is. Írassa ki az egyedi elemeket és azok darabszámát.

### **Megoldás**

Set: halmaz. Mint a matematikai halmazban, egy elem csak egyszer szerepel.

Több fajta Set van. Ezek közül egyik a HashSet. [https://www.w3schools.com/java/java\\_hashset.asp](https://www.w3schools.com/java/java_hashset.asp)

Az belső u.n. hash tábla segítségével gyorsan lehet vele műveleteket végrehajtani.

```
package csomag1;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.HashSet;
import java.util.Scanner;
import java.util.Set;

public class Feladat {

 Feladat() throws FileNotFoundException{
 Megold();
 }

 void Megold() throws FileNotFoundException{
 Scanner olvas = new Scanner(new File("adatok.dat"));
 Set<Integer>halmaz = new HashSet<Integer>();
 // Így is jó: Set<Integer>halmaz = new HashSet<>();
 while(olvas.hasNext())
 halmaz.add(Integer.parseInt(olvas.nextLine()));
 for(int szam:halmaz)
 System.out.println(szam);
 System.out.println("Az egyedi elemek száma="+halmaz.size());
 }

 public static void main(String[] args) throws FileNotFoundException {
 Feladat feladat = new Feladat();
 }
}
```

### **3. feladat – Fájlból olvasás, List**

A számok.txt fájlban valós számok vannak egymás alatt. A számok beolvasása után határozza meg azok minimumát és írassa ki. Írassa ki a számokat fordított sorrendben egymás mellé a képernyőre.

### **Megoldás**

Mivel itt lehetnek azonos számok, azért nem érdemes halmazban tárolni azokat. Tömbben sem lenne érdemes tárolni, mert nem tudjuk, hogy hány szám van a fájlban. A lista jó lesz, mert bármikor új elemet adhatunk hozzá.

List: lista. Több fajta lista van. Ezek egyike az ArrayList

```
package csomag1;
```

```

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;

public class Feladat {

 Feladat() throws FileNotFoundException{
 Megold();
 }

 void Megold() throws FileNotFoundException{
 Scanner olvas = new Scanner(new File("adatok.dat"));
 List<Double> lista = new ArrayList<>();
 while(olvas.hasNext())
 lista.add(Double.parseDouble(olvas.nextLine()));
// ciklussal is meghatározhatnánk:
 System.out.println("A lista legkisebb száma="+Collections.min(lista));
 System.out.println("A lista elemei fordított sorrendben:");
 for(int i=lista.size()-1;i>=0;i--)
 System.out.print(lista.get(i)+" ");
 }

 public static void main(String[] args) throws FileNotFoundException {
 Feladat feladat = new Feladat();
 }
}

```

#### 4. feladat – Fájlból olvasás, Map

Dolgozatok eredményéről statisztikát kell készíteni.

Az eredményeket(egészek) a jegyek.txt fájlban tároljuk egymás alatt. Készítsen programot, amely összeszámolja az 1-es, 2-es stb. dolgozatokat. Megszámolja a jegyeket és kiszámolja azok átlagát. Az eredményeket kiírja a képernyőre.

#### Megoldás

Map: kulcs-érték párokat tartalmaz. A Map nem tud tárolni duplikált kulcsokat, egy kulcshoz csak egy érték rendelhető.

HashMap: A Map-nak több fajtája lehet. Ezek közül az egyik a HashMap. Mint a HashSet-nél gyors elérést tesz lehetővé egy belső hash-tábla segítségével.

```

package csomag1;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class Feladat {

```

```

Feladat() throws FileNotFoundException{
 Megold();
}

void Megold() throws FileNotFoundException{
 Scanner olvas = new Scanner(new File("jegyek.txt"));
 Map<Integer, Integer> jegyek = new HashMap<Integer, Integer>();
 int jegy, összeg=0, db = 0;
 while(olvas.hasNextLine()){
 jegy = Integer.parseInt(olvas.nextLine());
 összeg += jegy;
 db++;
 if(!jegyek.containsKey(jegy))
 jegyek.put(jegy, 1);
 else
 jegyek.put(jegy, jegyek.get(jegy)+1);
 }
 System.out.println("Jegyek és azok darabszáma: "+jegyek);
 System.out.println("Két oszlopban:");
 for(int j:jegyek.keySet())
 System.out.println(j+"\t"+jegyek.get(j));
 System.out.println("Jegyek átlaga= "+1.0*összeg/db);
}

public static void main(String[] args) throws FileNotFoundException {
 Feladat feladat = new Feladat();
}
}

```

### Split - String darabolása

Pl. String str = „egy kettő három négy öt”;

String[] strTomb = str.split(" ");

5 elemű String tömböt készít az str Stringet a                      szöközők mentén darabolva.

## Összetett feladatok - Array, Set, List, Map, Class

### 5. feladat – Fájlból olvasás, Map

Hozzuk létre a **csomag1** csomagot és benne a **Feladat** osztályt a **main()** metódussal.

Egy cégnél a csoportok egy raktárban adják le az adott nap elkészített termékeiket. Nem minden csoport dolgozik minden nap. Egy szövegfájlban (csoportok.txt) két oszlopban tároljuk a csoportok által leadott termékék számát. A két oszlopot tabulátor választja el egymástól. Első oszlop a csoport azonosítója (szöveg), második oszlop a termékek száma (egész).

A szövegfájl adatait feldolgozva írassa ki, hogy az adott időszakban:

- hány csoport adott le terméket
- melyik csoport hány terméket adott le (két oszlopban)
  - Ezt a kiíratást három formában adja meg:
    - rendezés nélkül
    - A csoportok neve szerint csökkenő sorrendben
    - A leadott termékek szerint növekvő sorrendben
- melyik csoport adta le a harmadik legtöbb terméket?

### Megoldás

IntelliJ-ben:



- New Project
- src mappán állva: File menü / New / Package: csomag1
- File menü / New / Java Class: Feladat
- beírni: main + Enter => kiegészíti a main() metódussal

package csomag1;

// Az import-okat nem nekünk kell beírni, az IntelliJ beírja

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;
import java.util.TreeMap;
import java.util.TreeSet;
```

```
public class Feladat {
```

// throws FileNotFoundException: nem kell magunknak beírni, IntelliJ beírja

```
Feladat() throws FileNotFoundException{
 Megold();
}
```

```
void Megold() throws FileNotFoundException{
 Scanner olvas = new Scanner(new File("csoportok.txt"));
 Map<String, Integer> csoportok = new HashMap<String, Integer>();
 String sor;
 String csoport;
 int db;
 while(olvas.hasNextLine()){
 sor = olvas.nextLine();
 csoport = sor.split("\t")[0];
 db = Integer.parseInt(sor.split("\t")[1]);
 if(!csoportok.containsKey(csoport))
 csoportok.put(csoport, db);
 else
 csoportok.put(csoport, csoportok.get(csoport)+db);
 }
 System.out.println("Csoportok száma="+csoportok.keySet().size());

 System.out.println("Nem rendezve:");
 for(String csop:csoportok.keySet())
 System.out.println(csop+"\t"+csoportok.get(csop));
 System.out.println();
```

//Helyfoglalás szempontjából nem ez a legjobb megoldás. Lásd CompareTo következő fejezet.

// De ez egyszerűbb. Kiseb adathalmazokra elég.

```
System.out.println("Csoportok szerint rendezve:");
Map<String, Integer> csoportok2 = new TreeMap<String, Integer>(csoportok);
for(String csop:csoportok2.keySet())
 System.out.println(csop+"\t"+csoportok2.get(csop));
System.out.println();
```

```

 System.out.println("Darabszámok szerint rendezve:");
 Set<Integer> dbRendezett = new TreeSet<Integer>(csoportok.values());
 for(int darab:dbRendezett)
 for(String csop:csoportok.keySet())
 if(csoportok.get(csop) == darab)
 System.out.println(csop+"\t"+csoportok.get(csop));
 System.out.println();
 }

 public static void main(String[] args) throws FileNotFoundException {
 Feladat feladat = new Feladat();
 }
}

```

## 6. feladat – Fájlból olvasás, Class, List

Hozzuk létre a **feladat** csomagot és benne a **Feldolgoz** osztályt a **main()** metódussal.

A **szemelyek.txt** fájlban személyek adatai vannak. Az adatok pontosvesszővel vannak elválasztva egymástól. Az első sor tartalmazza a mezőneveket.

### A fájl első néhány sora:

```

név;cím;település;kor;magasság;súly
Kovács Ferenc;Kecskemét;35;182;78.3
Nagy Tibor;Debrecen;27;178;73.4
Kovács Éva;Szeged;32;179;82.7

```

- nem tudjuk előre, hogy hány sor van a fájlban.
- A fájlban lehetnek felesleges üres sorok is.
- A sorok végén lehetnek felesleges szóközök is.

Egy személy adatának tárolásához hozzunk létre egy **Személy** osztályt. A feltöltött osztálypéldányokat egy **személyek** listában tároljuk.

Olvassuk be az adatokat a fájlból a listába. Adjuk még a listához a következő személy adatait:

```
Tóth Judit;Győr;37;175;78.3
```

Írassuk ki az életkorok átlagát és azon személyek nevét és életkorát, akik idősebbek, mint az átlag.

```

korÁtlag=32.75
Kovács Ferenc 35
Tóth Judit 37

```

## Megoldás

A későbbiekben az osztályok változói **private**-ok lesznek, setter és getter metódusokkal fogjuk elérni azokat. Itt az osztályt csak adattárolásra használjuk, ezért az egyszerűség miatt közvetlenül érjük el adatait.

```
package feladat;
```

```

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

```

```

class Személy {
 String név;
 String település;
 int kor;
 int magasság;
 double súly;
}

public class Feldolgoz {
 List<Személy> személyek =new ArrayList<Személy>();
 Feldolgoz() throws FileNotFoundException {
 Feltölt();
 Feldolgoz();
 }
 void Feltölt() throws FileNotFoundException {
 Scanner olvas = new Scanner(new File("személyek.txt"));
 String sor;
 String[] stringTömb;
 Személy személy;
 int db=0;
 while(olvas.hasNextLine()){
 sor = olvas.nextLine().trim();
 db++;
 if(db>1 && !sor.equals("")){
 stringTömb = sor.split(";");
 személy = new Személy();
 személy.név = stringTömb[0];
 személy.település = stringTömb[1];
 személy.kor = Integer.parseInt(stringTömb[2]);
 személy.magasság = Integer.parseInt(stringTömb[3]);
 személy.súly = Double.parseDouble(stringTömb[4]);
 személyek.add(személy);
 }
 //System.out.println(sor);
 }
 olvas.close();
 személy = new Személy();
 személy.név = "Tóth Judit";
 személy.település = "Győr";
 személy.kor = 37;
 személy.magasság = 175;
 személy.súly = 78.3;
 személyek.add(személy);
 }

 void Feldolgoz(){
 double korÁtlag=0;
 for(Személy személy:személyek)
 korÁtlag+=személy.kor;
 korÁtlag /= személyek.size();
 System.out.println("korÁtlag="+korÁtlag);
 for(Személy személy:személyek)

```

```

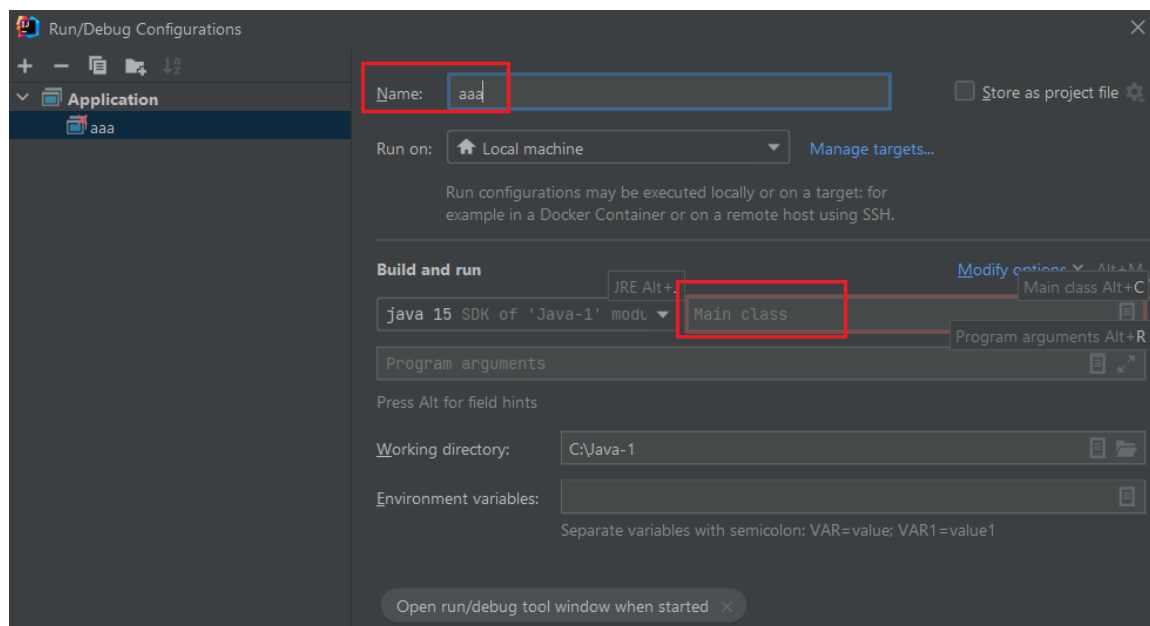
 if(személy.kor>korÁtlag)
 System.out.println(személy.név+"\t"+személy.kor);
 }

 public static void main(String[] args) throws FileNotFoundException {
 Feldolgoz feldolgoz = new Feldolgoz();
 }
}

```

**Ha a futtatási zöld gomb nem aktív:**

Run menü / Edit configurations / Add new run configuration / Application



A Main class-nál meg kell adni, hogy melyik osztály tartalmazza a main() metódust.

Belépve ebbe a választásba:

Várni kell, amíg a program megkeresi az osztályokat a project-ben. A felajánlott **Feldolgoz** osztályt kell kiválasztani:

## 7. feladat – Fájlból olvasás, List<List<Integer>>

Az adatok.txt fájlban egész számokat tárolunk több sorban. Minden sorban több szám lehet, egymástól szóközzel elválasztva.

Az adatok beolvasása után írassa ki a képernyőre és a kimenet.txt fájlba:

- sorok száma
- számok átlaga
- soronként a számok összege
- legkisebb szám

Alakítsa ki a következő mappaszerkezetet:

### mappa1

Foprogram.java

#### adattár

Tarolo.java

#### feldolgozó

Feldolgoz.java

## Megoldás

### Tarolo.java

```
package mappal.adattar;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

// Tarolo osztály az adatok tárolására + Beolvasás
public class Tarolo {
 // Listát tartalmazó lista
 private List<List<Integer>> adatok = new ArrayList<List<Integer>>();

 public void Beolvas(String fajlnev) throws FileNotFoundException{
 Scanner beolvas = new Scanner(new File(fajlnev));
 String[] stringTomb;
 List<Integer> szamok;
 String sor;
 while(beolvas.hasNextLine()){
 sor = beolvas.nextLine();
 // üres és csak szóközőket tartalmazó sorok nem kellenek
 if(!sor.trim().equals("")){
 // Szóközők mentén daraboljuk a sort:
 stringTomb = sor.split(" ");
 szamok = new ArrayList<Integer>();
 // Betöltjük a a sor számait egy listába
 for(String str:stringTomb)
 szamok.add(Integer.parseInt(str));
 // A belső listát betöltjük a fő listába
 adatok.add(szamok);
 }
 }

 // A lista lekérése
 public List<List<Integer>> getAdatok(){
 return adatok;
 }
 }
}
```

### Feldolgoz.java

```
package mappal.feldolgozo;

import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Collections;
```

```
import java.util.List;
```

```
// Feldolgoz osztály
```

```
public class Feldolgoz {
 private Tarolo tarolo = new Tarolo();
 public Feldolgoz() throws FileNotFoundException {
 // Az adatok beolvasása fájlból
 tarolo.Beolvas("adatok.txt");
 Feldolgozás();
 }

 void Feldolgozás() throws FileNotFoundException {
 PrintWriter kiir = new PrintWriter("kimenet.txt ");
 List<List<Integer>> adatok = tarolo.getAdatok();
 // Sorok száma
 String szoveg = "Sorok száma =" + adatok.size();
 String szoveg2 = "\nSorok összege:\n";
 // monitorra és fájlba is kiírja
 System.out.println(szoveg); kiir.println(szoveg);
 double osszeg=0;
 int db=0, sorOsszeg;
 List<Integer> legkisebbek = new ArrayList<Integer>();
 // végig megy a sorokon
 for(List<Integer> lista:adatok){
 sorOsszeg = 0;
 for(int szam:lista){
 // számok összege: átlaghoz
 osszeg +=szam;
 // sorok összege
 sorOsszeg += szam;
 }
 // sorok összege egy string-ben
 szoveg2 += sorOsszeg+"\n";
 // összes darabszám: átlaghoz
 db += lista.size();
 // soronként legkisebbek
 legkisebbek.add(Collections.min(lista));
 }
 szoveg = "A számok átlaga=" + (osszeg/db);
 // monitorra és fájlba is kiírja
 System.out.println(szoveg); kiir.println(szoveg);
 System.out.println(szoveg2); kiir.println(szoveg2);
 szoveg = "A legkisebb érték=" + Collections.min(legkisebbek);
 System.out.println(szoveg); kiir.println(szoveg);
 kiir.close();
 }
}
```

Foprogram.java

```
package mappal;
```

```
import java.io.FileNotFoundException;
```

```
import mappa1.feldolgozó.Feldolgoz;
```

```
// Foprogram osztály
```

```
public class Foprogram {
 Foprogram() throws FileNotFoundException {
 Feldolgoz feldolgoz = new Feldolgoz();
 }

 public static void main(String[] args) throws FileNotFoundException {
 Foprogram foFoprogram = new Foprogram();
 }
}
```

Ha van még idő az órából, akkor nézzék átgyakorlásképpen a kihagyott 1-4 feladatokat is.

## 2. gyakorlat – Objektum orientált programozás

### 1. feladat - Film

Hozzon létre egy mappát a projektnek pl. **c:\Java-Film**, amibe készítse el a **Filmek** projektet.

Hozza létre a **filmek** csomagot.

Készítsen egy **Film osztályt (Film.java)**, amely egy film cím, hossz (percekben) adatait tartalmazza.

Lehet inicializálni (konstruktor), megadva ezeket az adatokat. Meg lehessen adni az adatokat külön-külön is.

Ha a megadott cím egy üres String, akkor a cím legyen "Ismeretlen". Ha a megadott hossz negatív, akkor a hossz legyen 0.

Meg lehet kapni az adatait egy Stringben (*cím,hossz perc*) alakban.

Készítsen egy **futtatható osztályt(main metódus!)** (**Feladat.java**), amelyben beolvassa a Források közül a filmek adatait.

Az adatok vesszővel vannak elválasztva egymástól. Az első sor tartalmazza a mezőneveket.

#### A fájl első néhány sora:

*cím,hossz*

Saul fia,123

Garfield,69

Óz,144

Gyűrűk Ura 3,201

- nem tudjuk előre, hogy hány sor van a fájlban.
- A fájlban lehetnek felesleges üres sorok is.
- A sorok végén lehetnek felesleges szóközök is.

Írja még hozzá a következő filmet:

Hobbit 1, 211

Írassa ki a filmek adatait.

- Keresse meg a leghosszabb filmet és adja meg.
- Írassa ki a filmek neveit növekvő sorrendben.

### Megoldás

Az osztályokat lehet egy fájlba is írni, vagy külön-külön fájlokba. Nagyobb programnál, ami több osztályt tartalmaz jobb minden osztályt külön fájlba írni, itt is úgy csináljuk. A fájl neve és az osztály neve megegyezik. Például a Film osztály a Film.java fájlban van.

#### **Film.java**

```
package filmek;
```

```
public class Film {
```

```
 private String cím;
```

```
 private int hossz;
```

```
 public Film(String cím, int hossz) { // A Film osztály konstruktora
```

```
 // Ha a megadott cím üres string
```

```
 if (cím.length() == 0)
```

```
 this.cím = "Ismeretlen";
```

```
 else
```

```
 this.cím = cím;
```

```
 // ha a megadott hossz negatív
```



```

 if (hossz < 0)
 this.hossz = 0;
 else
 this.hossz = hossz;
 }
 // A cím megadása
 public void setCím(String cím) {
 if (cím.length() == 0)
 this.cím = "Ismeretlen";
 else
 this.cím = cím;
 }
 // A hossz megadása
 public void setHossz(int hossz) {
 if (hossz < 0)
 this.hossz = 0;
 else
 this.hossz = hossz;
 }
 @Override
 // Az osztály szöveges jellemzése (cím, hossz, perc) alakban
 public String toString() {
 return "(" + cím + ", " + hossz + " perc)";
 }
 //A cím lekérdezése
 public String getCím() {
 return cím;
 }
 //A hossz lekérdezése
 public int getHossz() {
 return hossz;
 }
}

```

### **Feladat.java**

```

package filmek;
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;
public class Feladat {
 // A Feladat osztály konstruktora
 Feladat() throws FileNotFoundException {
 Feltölt();
 Feldolgoz();
 }
 List<Film> filmek = new ArrayList<Film>();
 void Feltölt() throws FileNotFoundException {
 Scanner olvas = new Scanner(new File("filmek.txt"));
 String sor;
 String[] stringTömb;
 int db=0;
 while(olvas.hasNextLine()){
 sor = olvas.nextLine().trim();

```

```

 db++;
 if(db>1 && !sor.equals("")){
 stringTömb = sor.split(",");
 filmek.add(new Film(stringTömb[0], Integer.parseInt(stringTömb[1])));
 }
 }
 filmek.add(new Film("Hobbit 1", 211));
 // filmek adatainak kiírása
 for (Film film : filmek)
 System.out.println(film);
}

void Feldolgoz(){
 // leghosszabb film megkeresése és kiírása
 Film leghosszabb = filmek.get(0);
 for (Film film : filmek)
 if (film.getHossz() > leghosszabb.getHossz())
 leghosszabb = film;
 System.out.println("A leghosszabb: " + leghosszabb);
 // filmek nevei növekvő sorrendben:
 Set<String> nevek = new TreeSet<String>();
 for (Film film : filmek)
 nevek.add(film.getCím());
 System.out.println(nevek);
}

public static void main(String[] args) throws FileNotFoundException {
 // az osztály példányosítása
 Feladat feladat = new Feladat();
}
}

```

## 2. feladat - Tanuló

Hozzon létre egy mappát a projektnek pl. **c:\Java-Tanuló**, amibe készítse el a **Tanulok** projektet.

Hozza létre a **tanulók** csomagot.

Készítsen egy **Tanuló osztályt (Tanuló.java)**, amely egy tanuló nevét és 10 jegyét tárolja.

JEGYEKSZÁMA = 10: final, az osztály saját eleme(static).

A 10 jegyet egy tömbben tároljuk.

Jegyek tartománya: 0-5, azaz lehessen 0 is.

0: határozatlan: a számításoknál (pl. jegyek száma, átlaga) kihagyjuk.

LEGNAGYOBBJEGY = 5: final, az osztály saját eleme.

(int LEGNAGYOBBJEGY = 5)

- Lehessen inicializálni megadva a tanuló nevét (a jegyeket nullával inicializálja).  
A Java alaptól 0-ra állítja
- Lehessen megadni a tanuló egy adott indexű jegyét.  
Ha az  $0 \leq \text{index} \leq \text{JEGYEKSZÁMA}$  és  $0 \leq \text{jegy} \leq \text{LEGNAGYOBBJEGY}$ , akkor beírja a jegyet, különben nem. A metódus visszatérési értéke legyen igaz, ha beírta a jegyet, különben hamis.
- Lehessen megkapni a jegyek számát. (nem nulla jegyek száma)
- Lehessen meghatározni a tanuló átlagát csak a nem nulla jegyeket figyelembe véve.
- Lehessen megkapni a tanuló jegyeit egy String-ben szóközzel elválasztva
- Lehessen stringben megkapni a tanuló nevét, jegyeinek számát, jegyeit és átlagát.
- Lehessen megkapni a tanuló egy adott jegyét

Készítsen egy **futtatható osztályt (Feladat.java)**, amelyben létrehoz két tanulót beolvasva a nevüket a billentyűzetről és véletlenszerűen generálja a jegyeiket. Majd írja ki mindkét tanuló adatait és hogy melyiknek jobb az átlaga.

## Megoldás

### **Tanuló.java**

```
package tanuló;
```

```
public class Tanuló {
```

```
 private static final int JEGYEKSZÁMA = 10;
```

```
 private String név;
```

```
 private final int[] jegyek = new int[JEGYEKSZÁMA];
```

```
 private static final int LEGNAGYOBBJEGY = 5;
```

```
 // Tanuló osztály konstruktora
```

```
 public Tanuló(String név) {
```

```
 this.név = név;
```

```
 }
```

```
 // adott indexű jegy beírása
```

```
 public boolean jegyetAd(int hova, int mit) {
```

```
 // csak 0.. LEGNAGYOBBJEGY értékű jegyet fogad el.
```

```
 if (hova >= 0 && hova < JEGYEKSZÁMA && mit >= 0 && mit <= LEGNAGYOBBJEGY) {
```

```
 jegyek[hova] = mit;
```

```
 return true;
```

```
 }
```

```
 return false;
```

```
 }
```

```
 // Nem 0 értékű jegyek számának megadása
```

```
 public int jegyekSzama() {
```

```
 int jegyekSzama = 0;
```

```
 for (int jegy: jegyek)
```

```
 if (jegy > 0)
```

```
 jegyekSzama++;
```

```
 return jegyekSzama;
```

```
 }
```

```
 // Átlag kiszámítása a nem 0 értékű jegyekre
```

```
 public double atlag() {
```

```
 int jegyekSzama = 0;
```

```
 double osszeg = 0.0;
```

```
 for (int jegy: jegyek)
```

```
 if (jegy > 0) {
```

```
 jegyekSzama++;
```

```
 osszeg += jegy;
```

```
 }
```

```
 if(jegyekSzama>0)
```

```
 return osszeg / jegyekSzama;
```

```
 else
```

```
 return 0;
```

```
 }
```

```
 // Az osztály jellemzése egy string-el
```

```
 @Override
```

```
 public String toString() {
```

```
 return "(" + név + ", jegyek száma: " + jegyekSzama() + " jegyei: " + jegyek() + ", átlaga: " + atlag() + ')';
```

```

 }
 // egy adott indexű jegy lekérdezése
 int adottJegy(int hányadik){
 return jegyek[hányadik];
 }
 // jegyek megadása egy string-ben szóközzel elválasztva
 String jegyek(){
 String str="";
 for(int jegy: jegyek)
 str +=jegy+" ";
 return str;
 }
}

```

### Feladat.java

```

package tanulók;
import java.util.Random;
import java.util.Scanner;
public class Feladat {
 // Feladat osztály konstruktora
 Feladat(){
 Megold();
 }
 void Megold(){
 Scanner beolv = new Scanner(System.in);
 // A két Tanuló példány létrehozása neveik beírásával
 System.out.println("Adja meg az első tanuló nevét");
 Tanuló t1 = new Tanuló(beolv.nextLine());
 System.out.println("Adja meg a második tanuló nevét");
 Tanuló t2 = new Tanuló(beolv.nextLine());
 // A két tanulónak véletlenszerűen generálja a jegyeit.
 Random vel = new Random();
 for(int i=0;i<10;i++){
 t1.jegyetAd(i, vel.nextInt(6));
 t2.jegyetAd(i, vel.nextInt(6));
 }
 // a két tanuló adatainak kiírása
 System.out.println("Első tanuló: "+t1);
 System.out.println("Második tanuló: "+t2);
 // kiírja melyiknek nagyobb az átlaga
 if(t1.átlag()>=t2.átlag())
 System.out.println("Az első átlaga nagyobb");
 else
 System.out.println("A második átlaga nagyobb");
 }
 public static void main(String[] args){
 Feladat feladat = new Feladat();
 }
}

```

### 3. feladat – Interface - metódus bemeneti paramétere interface

Készítsünk egy interfészt **ISzámok** néven.

Metódusa: `int összeg();`

Készítsünk két osztályt, ami implementálja az interfészt: **Egészek3**, **Lista**

**Egészek3:**

három változója van a következő értékekkel: 5, 10, 15

`összeg()` metódusa visszaadja a három változó összegét.

**Lista**

Tartalmaz egy egészekből álló listát, amit a példányosításkor töltünk fel a következő értékekkel: 3, 4, 5, 6

`összeg()` metódusa visszaadja a lista elemeinek az összegét.

A főosztályban

Készítsük el az `ÖsszegKíír` metódust, aminek bemeneti paramétere az **ISzámok** interfész.

Így a metódus meghívásakor minden olyan osztály lehet a bemeneti paramétere, ami implementálja az interfészt.

Kiírja az **összeg()** metódus értékét.

A **main()** metódusban példányosítsuk a **Egészek3** és a **Lista** osztályokat és hívjuk meg velük az `ÖsszegKíír` metódusokat.

### Megoldás

#### **Feladat.java**

```
import java.util.ArrayList;
import java.util.List;
interface ISzámok
{
 int összeg();
}
class Egészek3 implements ISzámok
{
 int a=5, b=10, c=15;
 public int összeg()
 {
 return a+b+c;
 }
}
class Lista implements ISzámok
{
 List<Integer> számokList = new ArrayList<Integer>();
 Lista(){
 számokList.add(3);
 számokList.add(4);
 számokList.add(5);
 számokList.add(6);
 }
 public int összeg()
 {
 int össz = 0;
 for(int szám:számokList)
 össz+=szám;
 return össz;
 }
}
```

```

 }
}
public class Feladat {
 static void ÖsszegKíír(ISzámok számok){
 System.out.println(számok.összeg());
 }
 public static void main(String[] args) {
 Egészek3 egészek3 = new Egészek3();
 ÖsszegKíír(egészek3);
 Lista lista = new Lista();
 ÖsszegKíír(lista);
 }
}

```

#### 4. feladat – Absztrakt osztály

Definiáljuk az Osztály1 absztrakt osztályt. Tagváltozója x, beolvas() metódusa absztrakt, kiír metódusa kiírja az x értékét.

A Feladat főosztályunkat származtassuk az Osztály1-ből. beolvas() metódusában olvassuk be x értékét a billentyűzetről. A main() metódusban példányosítsuk a főosztályt, konstruktorában hívjuk meg a beolvas() és a kiír() metódusokat.

**Megoldás** (nincs a megoldások.zip-ben)

##### **Feladat.java**

```

import java.util.Scanner;
abstract class Osztály1 {
 int x;
 public abstract void beolvas();
 public void kiír() {
 System.out.println("x= " + x);
 }
}
public class Feladat extends Osztály1 {
 Feladat(){
 beolvas();
 kiír();
 }
 public void beolvas() {
 Scanner olvas = new Scanner(System.in);
 System.out.println("Add meg x értékét!");
 x=olvas.nextInt();
 }
 public static void main(String[] args) {
 Feladat feladat = new Feladat();
 }
}

```

### 3. gyakorlat – JDBC – Adatbáziskezelés

Előadáson átnéztük a JDBC alapjait. Itt nem nézzük át újra, hanem példában alkalmazzuk.

Előadáson több módszert is néztünk JDBC feladatokhoz:

A, mellékelt JAR fájlal

B, Maven-el letölteni a JAR fájlt

C, Maven project-el, pom.xml fájlal

Itt az **A, mellékelt JAR fájlal** módszert nézzük meg. A további gyakorlatokon majd a **C, Maven project-el, pom.xml fájlal** módszert alkalmazzuk.

#### Feladat

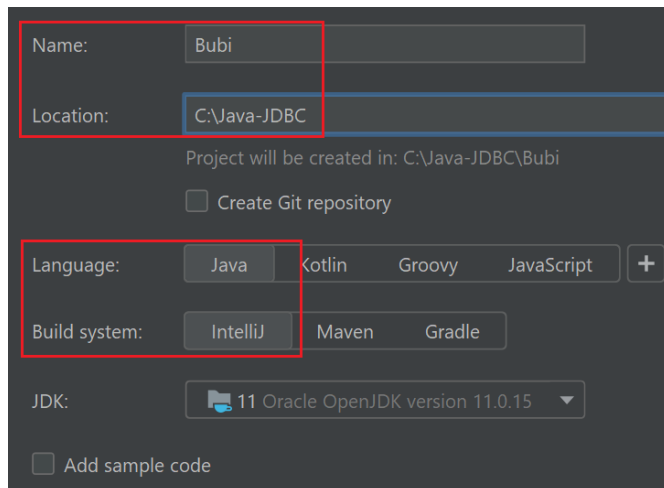
##### XAMPP és MySQL indítása

Források között lévő **bubi.sql** fájlból adatbázis importálása  
bubi adatbázis létrehozása  
gyujtoallomasok tábla létrehozása

Források mappában megtalálják a szükséges JDBC driver-t:  
**mysql-connector-java-5.0.8-bin.jar** (0,5MB)

Készítsünk egy mappát a projektnek. pl.: c:\Java-JDBC

IntelliJ-ben: Készítsük el a Java projektet a mappába pl. **Bubi** néven



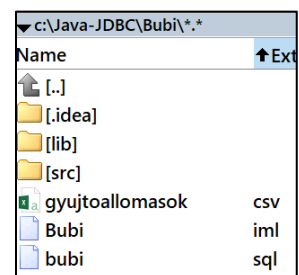
Készítsenek egy **lib** mappát az src mappa mellé és másolják be a **mysql-connector-java-5.0.8-bin.jar** fájlt a Források közül.

Nézzük meg, hogy ez valójában egy **ZIP** csomagolt fájl.

Nevezzük át mysql-connector-java-5.0.8-bin.**zip**-re

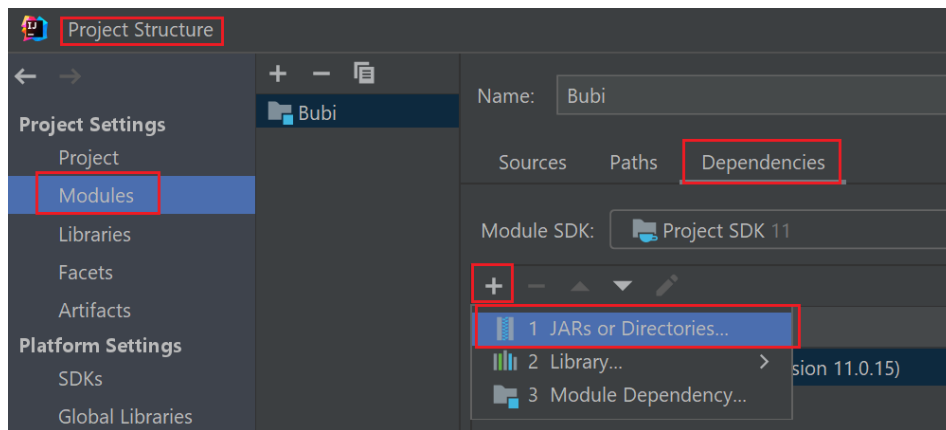
majd nevezzük vissza mysql-connector-java-5.0.8-bin.jar-ra

A források között találnak egy **gyujtoallomasok.csv** fájlt, amiben az adatbázis adatai találhatóak. Ezt is másolják az src mappa mellé.



##### Adjuk hozzá a JAR fájlt a projekthez:

File menü / Project structure / Modules / Dependencies / + / **JARs** or Directories...



Keressük meg és válasszuk ki a **mysql-connector-java-5.0.8-bin.jar** fájlt.

Készítsük el az **bubi** csomagot.

**Készítsük el a következő osztályokat, fájlokat:**

**Gyujtoallomas.java (modell osztály)**

Ezzel modellezzük az adatbázis tábla adatait.

**Modell osztályt** gyakran használnak az MVC mintánál: ez egy közbülső réteg az üzleti logika és a nézet között.

**Egy ilyen osztálypéldányban továbbítjuk az osztályok között egy-egy gyűjtőállomás adatait.**

Tagváltozók:

```
private int azonosito;
private String hely;
private int dokkoloMennyiseg;
private double gpsLon;
private double gpsLat;
```

- Készítsünk egy 5 paraméteres konstruktort az osztály tagváltozóinak inicializálásához.
- Készítsük el a Setter és Getter metódusokat a tagváltozókhoz.
- Készítsünk toString() metódust az osztály szöveges jellemzéséhez.

**BubiDbManager.java**

Ebben valósítjuk meg az adatbázis-műveleteket.

Metódusokat hozunk benne létre az adatbázis-műveletekhez:

**public List<Gyujtoallomas> getAll()**

Kigyűjti egy listába a gyűjtőállomásokat

**public int count()**

Megadja a tábla sorainak a számát.

**public int sum()**

Megadja a DokkoloMennyiseg értékek összegét

**public List<Gyujtoallomas> filter(int dokkoloMennyisegParam)**

Kigyűjti egy listába azokat a gyűjtőállomásokat, amelyiknél:

DokkoloMennyiseg > dokkoloMennyisegParam

Preparált műveletet használjon a dokkoloMennyisegParam értékéhez.

**public boolean insert(Gyujtoallomas gyujtoallomas)**

A paraméterként megadott gyűjtőállomás adatait betölti egy új adatbázis rekordba.

Preparált műveletet használjon mind az 5 értékhez.

A metódus visszatérési értéke legyen true, ha sikeres volt a művelet, különben false.

**Bubi.java (főosztály a main(...) metódussal )**



Készítse el a következő metódusokat:

**private static void filter(BubiDbManager manager, Scanner scanner)**

- Billentyűzetről beolvas egy dokkolómennyiséget a Scanner példány segítségével.
- A BubiDbManager példány filter metódusával kigyűjti azokat a gyűjtőállomásokat, ahol DokkoloMennyiség nagyobb mint a beolvasott érték.
- Kiírja a lista elemeit a képernyőre

**private static void insert(BubiDbManager manager, Scanner scanner)**

- Beolvas egy új gyűjtőállomás 5 adatát billentyűzetről
- A BubiDbManager példány segítségével egy új rekordot ad a táblához a beolvasott adatokkal
- A megoldásban nem ellenőrizzük, hogy van-e már a táblában a beolvasott azonosító (elsődleges kulcs). **A ellenőrzést valósítsa meg gyakorló házi feladatként.**

**public static void main(String[] args)**

- Készít 1-1 BubiDbManager és Scanner példányt.
- Kigyűjti az összes gyűjtőállomást egy listába
- Meghívja a következő metódusokat:  
Az első kettőnél kiírja az eredményt.

```
manager.count()
manager.sum();
filter(manager, scanner);
insert(manager, scanner);
```

**Készítsünk egy metódust, ami beolvassa a gyujtoallomasok.csv fájl adatait és betölti az adatbázisba.**

### Megoldás

**Gyujtoallomas.java (modell osztály)**

```
package bubi;
public class Gyujtoallomas {
 private int azonosito;
 private String hely;
 private int dokkoloMennyiseg;
 private double gpsLon;
 private double gpsLat;
```

**// A további metódusokat a a Code/Generate menüponttal készíttessük el:**

```
 public Gyujtoallomas(int azonosito, String hely, int dokkoloMennyiseg, double gpsLon, double
gpsLat) {
 this.azonosito = azonosito;
 this.hely = hely;
 this.dokkoloMennyiseg = dokkoloMennyiseg;
 this.gpsLon = gpsLon;
 this.gpsLat = gpsLat;
 }

 public int getAzonosito() {
 return azonosito;
 }

 public void setAzonosito(int azonosito) {
 this.azonosito = azonosito;
```

```

 }
 public String getHely() {
 return hely;
 }
 public void setHely(String hely) {
 this.hely = hely;
 }
 public int getDokkoloMennyiseg() {
 return dokkoloMennyiseg;
 }
 public void setDokkoloMennyiseg(int dokkoloMennyiseg) {
 this.dokkoloMennyiseg = dokkoloMennyiseg;
 }
 public double getGpsLon() {
 return gpsLon;
 }
 public void setGpsLon(double gpsLon) {
 this.gpsLon = gpsLon;
 }
 public double getGpsLat() {
 return gpsLat;
 }
 public void setGpsLat(double gpsLat) {
 this.gpsLat = gpsLat;
 }
 }
 @Override
 public String toString() {
 return "Gyujtoallomas{" + "azonosito=" + azonosito + ", hely=" + hely +
 ", dokkoloMennyiseg=" + dokkoloMennyiseg + ", gpsLon=" + gpsLon +
 ", gpsLat=" + gpsLat + '}';
 }
}

```

### **BubiDbManager.java**

```
package bubi;
```

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.ArrayList;
import java.util.List;

```

```

public class BubiDbManager {
 Statement statement;
 Connection connection;
 void Kacsolódás() throws SQLException {
 // characterEncoding=utf8: különben nem lesznek jók a magyar ékezetes karakterek:
 final String URL = "jdbc:mysql://localhost/bubi?user=root&characterEncoding=utf8";
 DriverManager.registerDriver(new com.mysql.jdbc.Driver());
 connection = DriverManager.getConnection(URL);
 }
}

```

```

 statement = connection.createStatement();
 }

 public List<Gyujtoallomas> getAll() {
 try {
 Kacsolódás();
 ResultSet resultSet = statement.executeQuery("SELECT * FROM Gyujtoallomasok");
 List<Gyujtoallomas> result = new ArrayList<>();
 while (resultSet.next()) {
 int az = resultSet.getInt("Azonosito");
 String hely = resultSet.getString("Hely");
 int dokkoloMennyiseg = resultSet.getInt("DokkoloMennyiseg");
 double gpsLon = resultSet.getDouble("GpsLon");
 double gpsLat = resultSet.getDouble("GpsLat");
 result.add(new Gyujtoallomas(az, hely, dokkoloMennyiseg, gpsLon, gpsLat));
 }
 return result;
 } catch (SQLException e) {
 System.out.println(e.getMessage());
 }
 return null;
 }

 public int count() {
 try {
 Kacsolódás();
 ResultSet resultSet = statement.executeQuery("SELECT COUNT(*) FROM Gyujtoallomasok");
 if(resultSet.next()) {
 int count = resultSet.getInt(1);
 return count;
 }
 } catch (SQLException e) {
 System.out.println(e.getMessage());
 }
 return -1;
 }

 public int sum() {
 try {
 Kacsolódás();
 ResultSet resultSet = statement.executeQuery("SELECT SUM(DokkoloMennyiseg) FROM Gyujtoallomasok");
 if(resultSet.next()) {
 int sum = resultSet.getInt(1);
 return sum;
 }
 } catch (SQLException e) {
 System.out.println(e.getMessage());
 }
 return -1;
 }

 public List<Gyujtoallomas> filter(int dokkoloMennyisegParam) {

```

```

try {
 Kacsolódás();
 PreparedStatement preparedStatement = connection.prepareStatement("SELECT * FROM
Gyujtoallomasok WHERE DokkoloMennyiseg > ?");
 List<Gyujtoallomas> result = new ArrayList<>();
 preparedStatement.setInt(1, dokkoloMennyisegParam);
 ResultSet resultSet = preparedStatement.executeQuery();
 while (resultSet.next()) {
 int az = resultSet.getInt("Azonosito");
 String hely = resultSet.getString("Hely");
 int dokkoloMennyiseg = resultSet.getInt("DokkoloMennyiseg");
 double gpsLon = resultSet.getDouble("GpsLon");
 double gpsLat = resultSet.getDouble("GpsLat");
 result.add(new Gyujtoallomas(az, hely, dokkoloMennyiseg, gpsLon, gpsLat));
 }
 return result;
} catch(SQLException e) {
 System.out.println(e.getMessage());
}
return null;
}

public boolean insert(Gyujtoallomas gyujtoallomas) {
 try {
 Kacsolódás();
 PreparedStatement preparedStatement = connection.prepareStatement("INSERT INTO
Gyujtoallomasok(Azonosito, Hely, DokkoloMennyiseg, GpsLon, GpsLat) VALUES (?, ?, ?, ?, ?)");
 preparedStatement.setInt(1, gyujtoallomas.getAzonosito());
 preparedStatement.setString(2, gyujtoallomas.getHely());
 preparedStatement.setInt(3, gyujtoallomas.getDokkoloMennyiseg());
 preparedStatement.setDouble(4, gyujtoallomas.getGpsLon());
 preparedStatement.setDouble(5, gyujtoallomas.getGpsLat());
 int rows = preparedStatement.executeUpdate();
 return rows == 1;
 } catch(SQLException e) {
 System.out.println(e.getMessage());
 }
 return false;
}
}

```

Minden metódusnál kapcsolódunk az adatbázishoz, ezért a metódusok végén érdemes lezárni a kapcsolatokat:

```

connection.close();
statement.close();
resultSet.close();

```

### **Bubi.java (main)**

```

package bubi;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.List;

```

```

import java.util.Scanner;

public class Bubi {

 public static void main(String[] args) throws FileNotFoundException {
 // TODO code application logic here
 BubiDbManager manager = new BubiDbManager();
 if(false) BeolvasásFájlbólBetöltésTáblába(manager);
 List<Gyujtoallomas> gyujtoallomasok = manager.getAll();
 // p: Egy módszer a System.out.println egyszerűsítésére:
 for (Gyujtoallomas gy : gyujtoallomasok)
 p(gy.toString());
 p("\nGyűjtőállomások száma: " + manager.count());
 p("\nÖsszes dokkolóhely: " + manager.sum());
 Scanner scanner = new Scanner(System.in);
 filter(manager, scanner);
 insert(manager, scanner);
 }

 static void BeolvasásFájlbólBetöltésTáblába(BubiDbManager manager) throws
 FileNotFoundException{
 Scanner beolv = new Scanner(new File("gyujtoallomasok.csv"), "UTF-8");
 String sor;
 int azonosito, dokkoloMennyiseg;
 String hely;
 double gpsLon, gpsLat;
 String[] stringTömb;
 while(beolv.hasNextLine()){
 sor=beolv.nextLine().trim();
 if(!sor.equals("") && !sor.startsWith("Azonosito;Hely;")){
 stringTömb = sor.split(";");
 azonosito = Integer.parseInt(stringTömb[0]);
 hely = stringTömb[1];
 dokkoloMennyiseg = Integer.parseInt(stringTömb[2]);
 gpsLon = Double.parseDouble(stringTömb[3]);
 gpsLat = Double.parseDouble(stringTömb[4]);
 manager.insert(new Gyujtoallomas(azonosito, hely, dokkoloMennyiseg, gpsLon, gpsLat));
 }
 }
 }

 private static void filter(BubiDbManager manager, Scanner scanner) {
 p("\nAdja meg a szükséges dokkolómennyiséget!");
 int dokkoloMennyiseg = Integer.parseInt(scanner.nextLine());
 List<Gyujtoallomas> gyujtoallomasok = manager.filter(dokkoloMennyiseg);
 for (Gyujtoallomas gy : gyujtoallomasok)
 p(gy.toString());
 }

 private static void insert(BubiDbManager manager, Scanner scanner) {
 p("\nA gyűjtőállomás felvételéhez adja meg az alábbi adatokat:");
 p("Azonosító:");
 int azonosito = Integer.parseInt(scanner.nextLine());
 }
}

```

```

p("Hely:");
String hely = scanner.nextLine();
p("Dokkoló mennyiség:");
int dokkoloMennyiseg = Integer.parseInt(scanner.nextLine());
p("GPS lon:");
double gpsLon = Double.parseDouble(scanner.nextLine());
p("GPS lat:");
double gpsLat = Double.parseDouble(scanner.nextLine());
Gyujtoallomas gyujtoallomas = new Gyujtoallomas(azonosito, hely, dokkoloMennyiseg, gpsLon,
gpsLat);
boolean result = manager.insert(gyujtoallomas);
if(result) {
 p("\nGyűjtőállomás hozzáadása sikeresen megtörtént!");
 List<Gyujtoallomas> gyujtoallomasok = manager.getAll();
 for (Gyujtoallomas gy : gyujtoallomasok)
 p(gy.toString());
} else
 p("\nHiba történt a gyűjtőállomás hozzáadása során...");
}
// Egy módszer a System.out.println egyszerűsítésére:
public static void p(String m) {
 System.out.println(m);
}
}

```

# Java Spring, Spring Boot, Bevezetés - Csak olvasmány

## Java Spring

<https://www.javatpoint.com/spring-tutorial>

[https://www.tutorialspoint.com/spring/spring\\_overview.htm](https://www.tutorialspoint.com/spring/spring_overview.htm)

<https://www.baeldung.com/spring-why-to-choose>

[https://en.wikipedia.org/wiki/Spring\\_Framework](https://en.wikipedia.org/wiki/Spring_Framework)

- A Spring egy nyílt forráskódú, inversion of controlt megvalósító Java alkalmazás keretrendszer. Az első változata 2003-ban jelent meg.
- A Spring keretrendszer magját képező szolgáltatásokat főként Java alkalmazás fejlesztésére használják a programozók, például web-alkalmazás fejlesztésére.
- Célja, hogy a nagyvállalati környezetbe szánt Java alkalmazások fejlesztését egyszerűbbé tegye.
- A korabeli Java EE szabvány pehelysúlyúbb alternatívájaként született.

## **Modules**

A Spring keretrendszer több önálló modulból épül fel, amelyek a következő szolgáltatásokat nyújtják a fejlesztők számára:

- **Inversion of control (IOC)** (kontroll megfordítása ) **konténer**: a Java objektumok életciklusának kezelése és az alkalmazás-komponensek testreszabása.
- **Aspektus orientált programozási** paradigma követésének lehetősége.  
AOP: egy magasabb szintű absztrakciót vezet be az OOP-hez képest, célja a modularitás növelése.
- **Adatelérés**: egyszerűsített JDBC, JPA, Data JPA, objektum-relációs lekérdezések (ORM), NoSQL
- **Tranzakciókezelés**: többféle tranzakció kezelő API-t tartalmaz.
- **Modell-nézet-vezérlő szabvány (MVC)**: egy HTTP- és servlet alapú keretrendszer segítségével valósítható meg, amelyet arra fejlesztettek ki, hogy bővíthetők és személyre szabhatóak legyenek a webszolgáltatások
- **Távoli eljáráshívás kezelő keretrendszer**: biztosítja a RPC alapú, hálózaton keresztül történő Java objektum importokat és exportokat. További támogatást nyújt a RMI, a CORBA és HTTP alapú protokollok használatára, beleértve a webszolgáltatásokat (SOAP) is.
- **Kötegelési eljárás** támogatása.
- **Biztonság (security)**
- **Azonosítás és azonosságkezelés**: biztonsági folyamatok konfigurálása, melyet a Spring projekthez tartozó, Spring Security alprojekt tesz lehetővé a különféle protokollok és módszerek biztosításával.
- **Üzenetkezelés**: általános üzenetkezelés továbbfejlesztése érhető el.
- **Tesztelés**: segítséget nyújt a unit- és az integrációs teszt írására.

## Java Spring Boot

<https://spring.io/projects/spring-boot>

[https://www.tutorialspoint.com/spring\\_boot/spring\\_boot\\_introduction.htm](https://www.tutorialspoint.com/spring_boot/spring_boot_introduction.htm)

<https://www.javatpoint.com/spring-boot-tutorial>

<https://www.baeldung.com/spring-boot>

<https://stackify.com/what-is-spring-boot/>

[https://www.ibm.com/cloud/learn/java-spring-boot#:~:text=Java%20Spring%20Boot%20\(Spring%20Boot,ability%20to%20create%20standalone%20applications](https://www.ibm.com/cloud/learn/java-spring-boot#:~:text=Java%20Spring%20Boot%20(Spring%20Boot,ability%20to%20create%20standalone%20applications)

Java Spring Boot (Spring Boot) is a tool that makes developing web application and microservices with Spring Framework faster and easier through three core capabilities:

- Autoconfiguration
- An opinionated approach to configuration
- The ability to create standalone applications

These features work together to provide you with a tool that allows you to set up a Spring-based application **with minimal configuration** and setup.

The biggest advantages of using Spring Boot versus Spring Framework alone are ease of use and faster development. In theory, this comes at the expense of the greater flexibility you get from working directly with Spring Framework.

Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

### Spring Boot Features

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration



## 4. gyakorlat - Egyszerű webalkalmazások - Spring

A Java Spring –el elsősorban web-es serveroldali alkalmazásokat készítünk, ezért felhasználjuk a Web-programozás-1 tárgynál tanult ismereteket.

### Feladat-1 - @SpringBootApplication, @Controller, Route, @GetMapping, @ResponseBody, JSON

Írjunk server oldali programot, ami

a, statikus weboldal: a <http://localhost:8080/feladat1a> URL címre kiírja az oldalra: Hello world! H1-es címsorral.

b, dinamikus weboldal: A kiírandó szövegen a felhasználó tevékenységétől függően változtatunk:

<http://localhost:8080/feladat1b?name=Egyetem> URL címre kiírja az oldalra: Hello Egyetem!

<http://localhost:8080/feladat1b> esetén ezt írja ki: Hello World!

c, Válaszként adjunk meg egy osztálypéldányt JSON formában. Restful API-nál fontos összetevő.

<http://localhost:8080/feladat1c>

### Megoldás

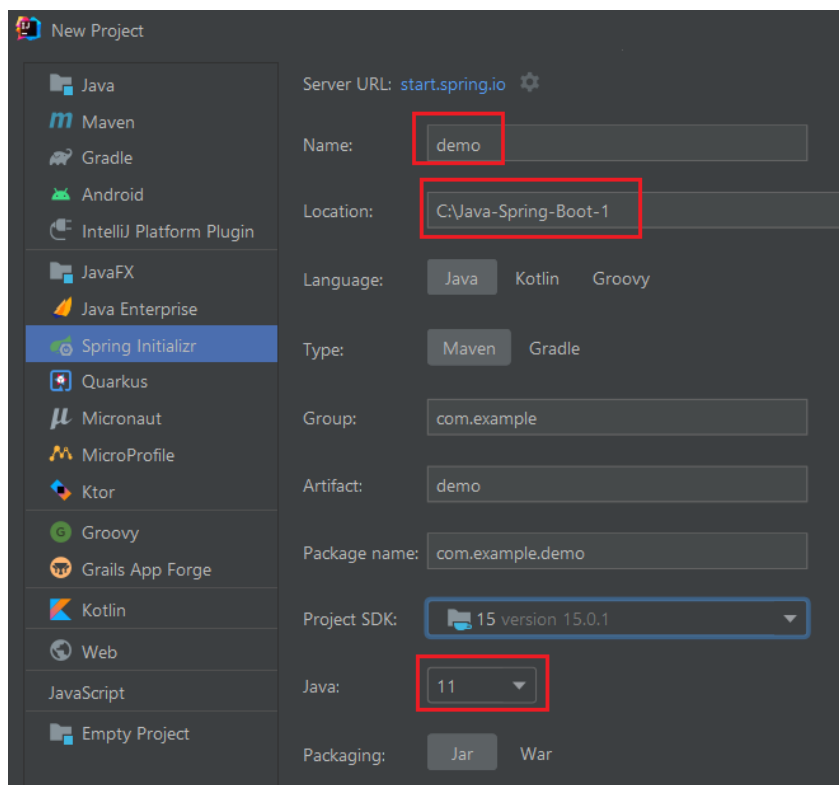
Készítsünk egy mappát a projektnek pl. **c:\Java-Spring-Boot-1**

### Előkészítés

Töltsük le az alaprendszert. Források.zip-be is bemásoltam.

#### A. módszer

IntelliJ / New project / Spring Initializr /



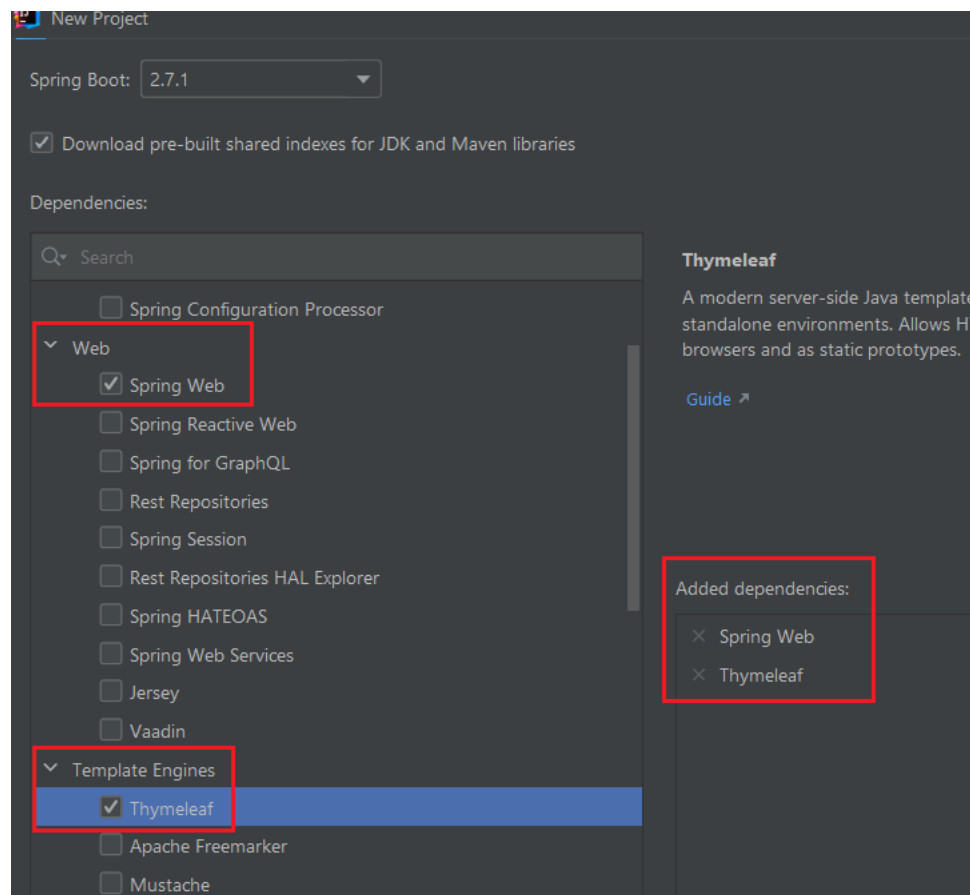
**Jó Java verziót válasszunk!**

Nézzük meg, hogy mi a Java verziószáma a gépen a következő módok valamelyikén:

- parancssorba: java -version
- Windows Kelékek / Vezérlőpult / Programok telepítése
- c:\Program Files\Java\ mappa tartalma

**Ha pl. 15-ös verziónk van, akkor ennél nagyobbat ne válasszunk, mert nem fog működni!**

Next

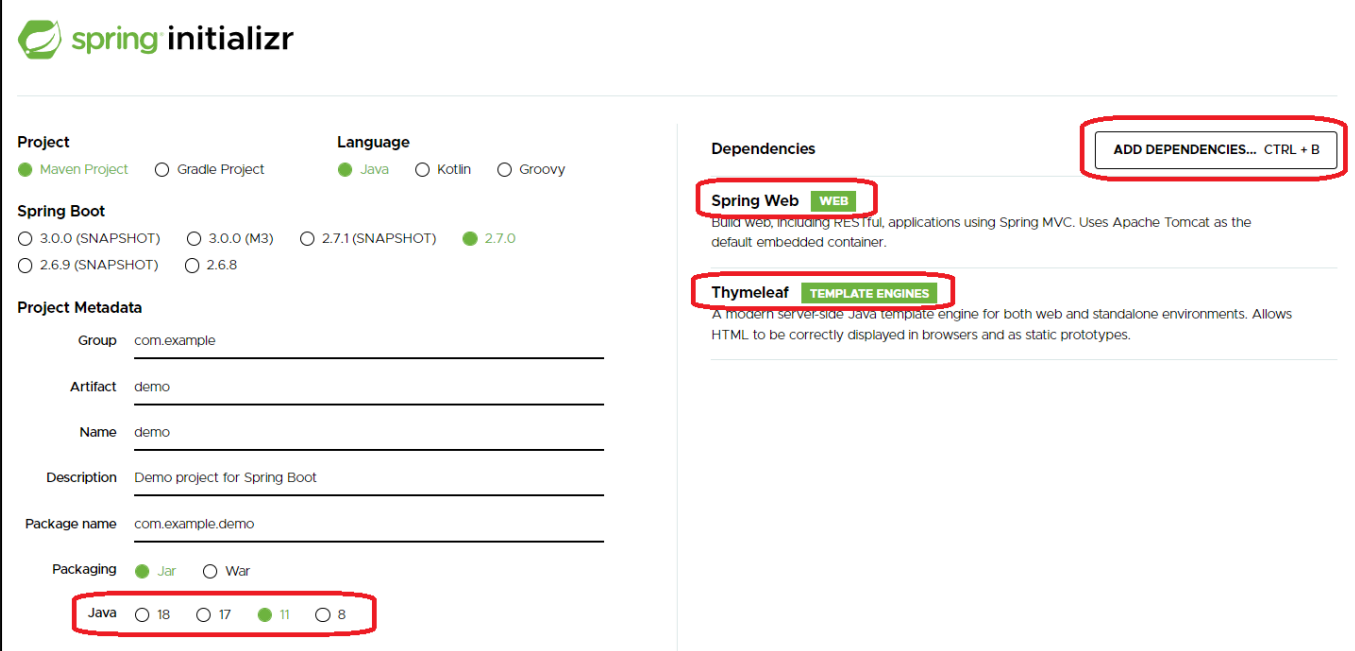


Finish

vagy:

**B, módszer: volt amikor hibát adott!**

<https://start.spring.io/>



The image shows the Spring Initializr web application interface. It is divided into three main sections: Project, Spring Boot, and Dependencies. The Project section includes options for Project (Maven Project selected), Language (Java selected), and Project Metadata (Group: com.example, Artifact: demo, Name: demo, Description: Demo project for Spring Boot, Package name: com.example.demo). The Spring Boot section includes options for Spring Boot version (2.7.0 selected) and Packaging (Jar selected). The Dependencies section includes a list of dependencies (Spring Web, Thymeleaf) and a button to add more dependencies. The Spring Web dependency is highlighted with a red box, and the Thymeleaf dependency is also highlighted with a red box. The Java version 11 is selected in the Project section, and the Spring Web dependency is highlighted with a red box. The Thymeleaf dependency is also highlighted with a red box. The ADD DEPENDENCIES... button is highlighted with a red box.

**Project**

☒ Maven Project ☐ Gradle Project

**Language**

☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**

☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M3) ☐ 2.7.1 (SNAPSHOT) ☒ 2.7.0

☐ 2.6.9 (SNAPSHOT) ☐ 2.6.8

**Project Metadata**

Group

Artifact

Name

Description

Package name

**Packaging**

☒ Jar ☐ War

**Java** ☐ 18 ☐ 17 ☒ 11 ☐ 8

**Dependencies**

**Spring Web** **WEB**

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Thymeleaf** **TEMPLATE ENGINES**

A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.

**ADD DEPENDENCIES... CTRL + B**

A **Spring Web** modult minden példánál fogjuk használni.

A **Thymeleaf** modul az első példákhoz nem kellene, majd csak a template-ek (nézetek) készítéséhez kell, hogy be tudjunk írni a HTML fájlba ehhez hasonló Thymeleaf kódokat:

**th:text="\${nev}**

<https://www.thymeleaf.org/>

=> GENERATE

Nézzük meg a **pom.xml** fájlt:

- mi a Java verziószáma: `<java.version>11</java.version>`
- milyen dependency-ket használ.

**spring-boot-starter-web, spring-boot-starter-test, spring-boot-starter-thymeleaf**

Az első megnyitásnál lassú (1-2 perc), mert letölti a szükséges dependency-ket.

**Ha a pom.xml fájlban pirossal bejelöl valamit**

```
pl. <parent>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-parent</artifactId>
 <version>2.7.0</version>
 <relativePath/> <!-- lookup parent from repository -->
</parent>
```

vagy a `src\main\java\com\example\demo\DemoApplication.java` -ban pirossal bejelöl valamit  
pl: `public static void main(String[] args) { .....`

akkor:

**File menü / Invalidate Caches**  
töröljük a Cache-t

**Volt néhány gépen (2 gépen a 15-ből), ahol a File menü / Invalidate Caches után is pirossal hibának jelölte ezt a plugin-t:**

```
<plugin>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

**ilyenkor segített:**

Adjuk hozzá a plugin verziót, ami megegyezik a spring boot verziójával:

```
<plugin>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-maven-plugin</artifactId>
 <version>${project.parent.version}</version>
</plugin>
```

<https://stackoverflow.com/questions/64639836/plugin-org-springframework-bootspring-boot-maven-plugin-not-found>

adding the **version of the plugin** in the pom.xml, **which is the same of spring boot's version**

A **\${project.parent.version}** helyett beírhatjuk a spring boot verziószámát is.

Ha ilyen hibaüzenetet kapunk futtatásnál:

```
error: release version 17 not supported

Language level is invalid or missing in pom.xml. Current project JDK is 15. Specify language level in demo
```

akkor lásd: **Jó Java verziót válasszunk!** résznél leírtakat.

**Ha nem aktív a Run gomb várakozási idő után sem:**

File menu / Invalidate Chaches ...

```

 /\ / ____ \ _ __| |__ \
 ()___ \| '_ \| |__ \| ___ \|
 \|___) | |_) | |__| |___) |
 '_____|_|_||_|_|_|_|_|_|_|
=====|_|=====|_|_|_|_|_|_|_|
:: Spring Boot :: (v2.5.4)

2021-09-15 17:45:42.095 INFO 5376 --- [main] com.example.demo.DemoApplication : Starting DemoApplication using Java 15.0.1 on DESKTOP-308FQ79 with PID 5376
2021-09-15 17:45:42.097 INFO 5376 --- [main] com.example.demo.DemoApplication : No active profile set, falling back to default profiles: default
2021-09-15 17:45:43.016 INFO 5376 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-09-15 17:45:43.024 INFO 5376 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-09-15 17:45:43.025 INFO 5376 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.52]
2021-09-15 17:45:43.096 INFO 5376 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-09-15 17:45:43.097 INFO 5376 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 932 ms
2021-09-15 17:45:43.449 INFO 5376 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context path ''
2021-09-15 17:45:43.463 INFO 5376 --- [main] com.example.demo.DemoApplication : Started DemoApplication in 1.919 seconds (JVM running for 3.387)

```

Az Apache Tomcat server egy webserverként működik és figyeli a 8080-as localhost portra jövő kéréseket.

Most még 404 -es hibaüzenetet kapunk, mert nincs hozzárendelve a "/" útvonalhoz semmi.

Az **src** mappa a program forráskódjának helye. A **target** mappába a lefordított osztályok kerülnek.

```
package com.example.demo;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication // kell
public class DemoApplication {
 public static void main(String[] args) {
 SpringApplication.run(DemoApplication.class, args);
 }
}
```

Automatikusan elvégzi a szükséges konfigurációkat. A main metódusban van a `SpringBootApplication.run` hívás.

a, statikus weboldal: a <http://localhost:8080/feladat1a> URL címre kiírja az oldalra: Hello world! H1-es címsorral.

```
.....
@SpringBootApplication
@Controller // kell
public class DemoApplication {
 public static void main(String[] args) {
 SpringApplication.run(DemoApplication.class, args);
 }
}
```

```

 }
 @GetMapping("/feladat1a")
 @ResponseBody // kell
 public String kiir1a() {
 return "<h1>Hello world!</h1>";
 }
}

```

A kódban pirossal jelzett hibáknál válasszuk a felajánlott **Import class** lehetőséget: beírja előre az importokat.

```

 @GetMapping("/feladat1a")
 @ResponseBody
 public String kiir1a() {
 return "<h1>Hello world!</h1>";
 }
}

```

A **@Controller** annotáció megjelöli az osztályt kontrollernek.

Konrollereknél általában a **@Controller** annotációt használjuk.

<https://www.baeldung.com/spring-controller-vs-restcontroller>

We can annotate classic controllers with the **@Controller** annotation.

We can use the **@Controller** annotation for traditional Spring controllers, and it has been part of the framework for a very long time.

Kivéve abban az esetben, amikor az osztály minden Mapping-jénél **@ResponseBody**-t használnánk, pl. egy Restful API-nál, mert ilyenkor a **@RestController** annotációt érdemes használni, akkor nem kell minden Mapping-hoz kiírni, hogy **@ResponseBody**.

**@RestController = @Controller + @ResponseBody**

<https://www.baeldung.com/spring-controller-vs-restcontroller>

Spring 4.0 introduced the **@RestController** annotation in order to simplify the creation of RESTful web services. It's a convenient annotation that combines **@Controller** and

**@ResponseBody**, which eliminates the need to annotate every request handling method of the controller class with the **@ResponseBody** annotation.

Itt nem lenne jó a **@RestController** annotáció, mert nem minden Mapping-nál használjuk a **@ResponseBody** annotációt (lásd lentebb).

**@ResponseBody**: amit a return-nél megadunk, azt adja vissza válaszul JSON formába átalakítva.

Ezzel ellentétben lesz majd az az eset, amikor pl. egy HTML fájlt (template) adunk vissza a böngészőnek (lásd lentebb). Ott a return-nél a template nevét adjuk meg pl. **return "szoveg"**; Itt nem a "szoveg" stringet adjuk meg válaszul, hanem a szoveg.html template-et. Ezért itt nem kell a **@ResponseBody**.

Ha itt is használnánk a **@ResponseBody**-t, akkor a "szoveg" stringet küldené válaszul a böngészőnek.

<https://www.baeldung.com/spring-request-response-body>

The **@ResponseBody** annotation tells a controller that the object returned is automatically serialized into JSON and passed back into the **HttpServletResponse** object.

**@GetMapping("/feladat1a")** (route, útvonal)

Web-programozásból tanulták a GET és a POST módszert, ez is hasonlót jelent. Lesz majd **@PostMapping** is.

A feladat1a route estén meghívja a kiir1a() metódust, ami visszatér a "Hello world!" stringet a hívó félnek. A szerver ezt a stringet küldi el a böngészőnek:

A kiir1a() metódust használja a <http://localhost:8080/feladat1a> címre küldött kérések válaszául.

**@GetMapping("/feladat1a")**

```

public String kiir1a() {
 return "Hello world!";
}

```

```
}
```

Futtassuk az alkalmazást.

<http://localhost:8080/feladat1a>

Kiírja: Hello world!

### **feladat1b**

b, dinamikus weboldal: A kiírandó szövegen a felhasználó tevékenységétől függően változtatunk:

<http://localhost:8080/feladat1b?name=Egyetem> URL címre kiírja az oldalra: Hello Egyetem!

<http://localhost:8080/feladat1b> esetén ezt írja ki: Hello World!

Folytassuk az előző kódot:

```
.....
 @GetMapping("/feladat1b")
 @ResponseBody
 public String kiir1b(@RequestParam(value = "name", defaultValue = "World") String name) {
 return String.format("Hello %s!", name);
 }
.....
```

@**RequestParam** annotáció azt mondja a Spring-nek, hogy várjon egy **name** értéket a kérésben. De ha nincs ilyen érték, akkor az alap “World” szót használja.

### **feladat1c**

c, Válaszként adjunk meg egy osztálypéldányt JSON formában. Restful API-nál fontos összetevő.

<http://localhost:8080/feladat1c>

Definiáljunk egy beágyazott osztályt (nested class) adatok néven.

Változói:

```
String név;
String cím;
int kor;
```

Készítsünk konstruktort és getter metódusokat.

A getter metódusok kellenek a helyes működéshez!

```
.....
class Adatok {
 String név;
 String cím;
 int kor;
 public Adatok(String név, String cím, int kor) {
 this.név = név;
 this.cím = cím;
 this.kor = kor;
 }
 public String getNév() {
 return név;
 }
 public String getCím() {
 return cím;
 }
 public int getKor() {
 return kor;
 }
}
```

```
}
}
```

**// @GetMapping("/feladat1b") alá:**

```
@GetMapping("/feladat1c")
@ResponseBody
public Adatok ClassKiirJSON() {
 Adatok adatok = new Adatok("Tóth Tibor", "Debrecen", 35);
 return adatok;
}
```

<http://localhost:8080/feladat1c>

```
{"név":"Tóth Tibor","cím":"Debrecen","kor":35}
```

**Már láttuk:**

<https://www.baeldung.com/spring-request-response-body>

The **@ResponseBody** annotation tells a controller that the object returned is automatically serialized into JSON and passed back into the `HttpServletResponse` object.

## Feladat-2 - olvasás HTML Template-ből

Írjunk szerver oldali programot, ami az oldal tartalmát a `szoveg.html` fájlból (template, nézet) olvassa be

<http://localhost:8080/feladat2>

<http://localhost:8080/feladat2>

**Hello, World!**

**Hello, World!**

**Hello, World!**

Készítsük el a template-et (nézet: HTML fájl): `szoveg.html`

Alapesetben a html file a `src/main/resources/templates` mappában legyen.

Más szerkesztővel is készíthető, pl. Jegyzettömb.

Vagy a file a `src/main/resources/templates` mappán állva: File menü / New /

`src/main/resources/templates/szoveg.html`

```
<!DOCTYPE HTML>
```

```
<html>
```

```
<head>
```

```
 <title>Getting Started: Serving Web Content</title>
```

```
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
```

```
</head>
```

```
<body>
```

```
 <h1>Hello, World!</h1>
```

```
 <h2>Hello, World!</h2>
```

```
 <h3>Hello, World!</h3>
```

```
</body>
```

```
</html>
```



## DemoApplication folytatása

```
.....
 @GetMapping("/feladat2")
 public String kiir1bFajlbol() {
 return "szoveg";
 }
```

!!! Nincs @ResponseBody

## Feladat-3 - HTML template módosítása futás közben - Model, Thymeleaf – MVC

MVC: Model-View-Controller (Modell-Nézet-Vezérlő)

Ebben a példában már lesz mind a három összetevő.

Írjunk szerver oldali programot, ami a szoveg2.html template-et módosítja futás közben és válaszként küldi azt.

```
src/main/resources/templates/szoveg2.html
<!DOCTYPE html>
<html lang="en">
<html xmlns:th="http://thymeleaf.org">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
 <p>Hello !</p>
</body>
</html>
```

th:text elemet HTML tag attribútumául lehet megadni.

## DemoApplication folytatása

```
.....
 @GetMapping("/feladat3")
 public String udvozlet(Model model) {
 model.addAttribute("nev", "Ferenc");
 model.addAttribute("magassag", 170);
 return "szoveg2";
 }
```

**Model osztály** A Model osztályt adattovábbításra használjuk.

Az MVC modellben a Model az adatforrás.

Hogy a modellt fel tudjuk használni, kell a **Model** paraméter:

```
public String greeting(Model model)
```

A Spring MVC-ben a **model** egy konténerként funkcionál, ami tartalmazza az alkalmazás adatait.

A model.addAttribute("nev", "Ferenc"); utasítással a "nev" attribútumhoz beírjuk a "Ferenc" paraméter értékét. A metódus paramétereként átadunk egy Model objektumot, abba feltöltjük az adatot és a nézetten – a sablonkezelővel – megjelenítjük.

**A Model-be tesszük azokat az attribútumokat, amiket fel akarunk használni a HTML fájlban.**

A feladathoz szükség van egy View technológiára (itt a **Thymeleaf**), ami biztosítja a HTML kód szerver oldali elkészítését.

A szoveg2.html fájlban a **Thymeleaf sablonkezelő rendszer illeszti be** a model példányból a "nev" attribútum értékét a  
`<p><strong>Hello </strong><span th:text="{nev}"> </span>!</p>`  
sorába.

<http://localhost:8080/feladat3>

**Hello** Ferenc!

**Nézzük meg a weboldal forrását a böngészőben:**

.....  
`<body>`  
    `<p><strong>Hello </strong><span>Ferenc</span>!</p>`  
`</body>`  
.....

## Spring alkalmazás futtatása IntelliJ nélkül – JAR fájl készítésével – egyelőre csak olvasmány

Készítünk egy JAR fájlt a projektből, amit már tudunk futtatni.

Az előző alkalmazást nyissuk meg IntelliJ-vel (Java-Spring-Boot-1).

Az IntelliJ ablak jobb szélén kattintsunk a Maven gombra:  
Erre megnyílik a Maven ablak.

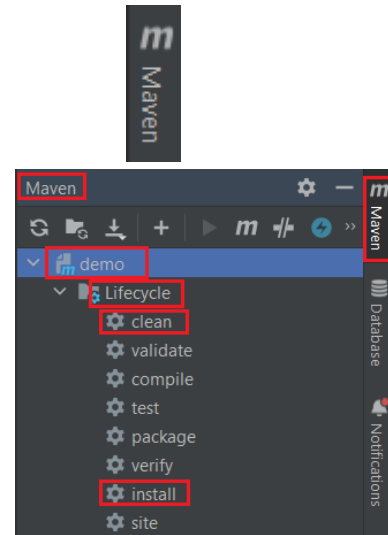
demo =>

Lifecycle=>

Először futtassuk a **clean**  
majd az **install** parancsot,  
kétszer kattintva rajtuk.

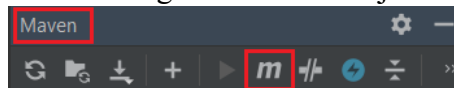
A **clean** törli a **target** mappát, ha van ilyen mappa.

Az **install** elkészíti a target mappát benne többek között a **demo-0.0.1-SNAPSHOT.jar** fájljal.



A **clean** és az **install** utasításokat az „Execute Maven Goal” gombnál is ki tudjuk adni:

**mvn clean install**



File menü / Close project

Nyissunk egy parancssort a projekt mappájában: c:\Java-Spring-Boot-1>

Adjuk ki a köv. utasítást: **java -jar target\demo-0.0.1-SNAPSHOT.jar**

Elindítja a szerveret.

Böngészőbe beírva: <http://localhost:8080/feladat1a>

# Hello world!

## 5. gyakorlat - 1. ZH a Spring előtti anyagból

## 6. gyakorlat - Spring Űrlapok

Az Űrlapok kezelése és a HTML kód a Web-programozás tárgy anyaga, itt csak felhasználjuk.

Feladat-1 - Űrlap, Model, @GetMapping, @PostMapping,

**Az űrlap induláskor:**

<http://localhost:8080/feladat>

**Az Űrlap adatainak megjelenítése:**

<http://localhost:8080/feladat2>

**Form**  
Id:   
Message:

**Result**  
id: 100  
content: Hello, World!  
[Submit another message](#)

Az Űrlap adatait **POST** módszerrel küldjük el a szervernek.

### Megoldás

Hozzunk létre egy mappát a projektnek, pl. **c:\Java-Spring-Boot-2**

### Előkészítés

Töltsük le az alaprendszert. **Források.zip**-be is bemásoltam.

**A, módszer**

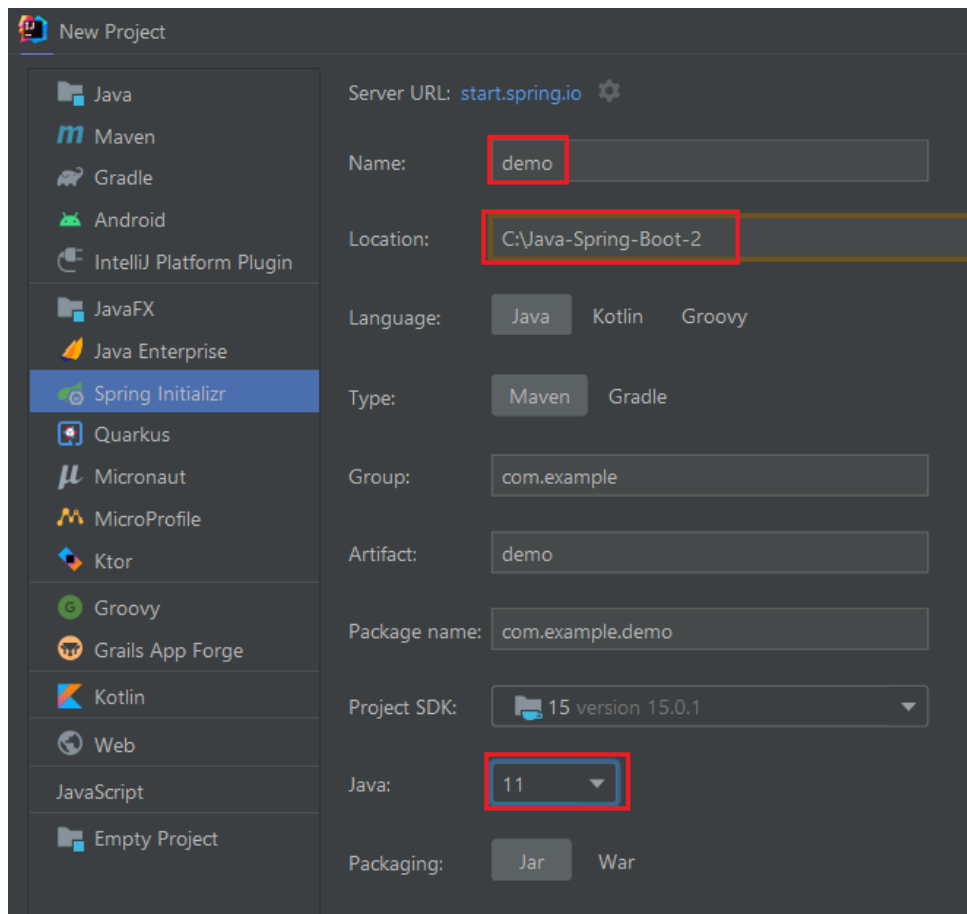
**IntelliJ / New project / Spring Initializr /**

**Dependencies**

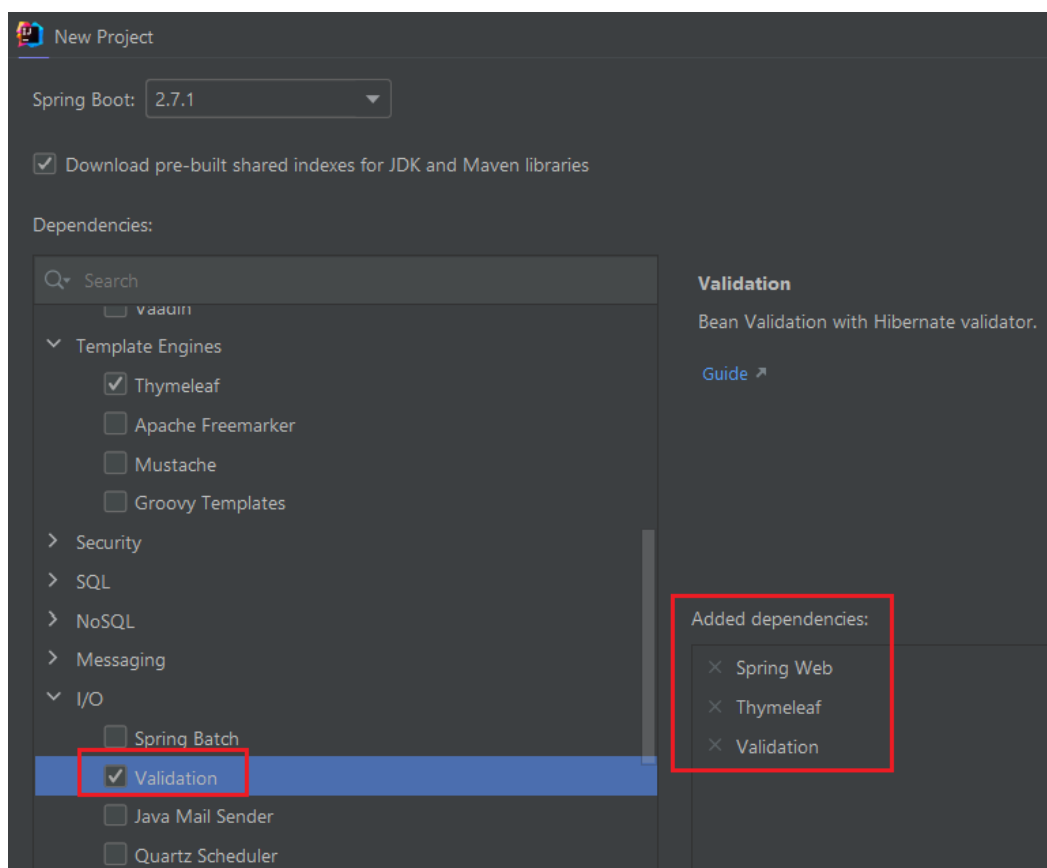
Már használtuk: Spring Web, Thymeleaf

**Validation:** A 2. feladathoz kell: Űrlap adatainak szerver oldali validációja.

Az alaprendszert a **Források.zip**-be is bemásoltam.



Next



Finish

vagy:

**B, módszer: volt amikor hibát adott!**

<https://start.spring.io/>

spring initializr

**Project**  
☒ Maven Project ☐ Gradle Project

**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M3) ☐ 2.7.1 (SNAPSHOT) ☒ 2.7.0  
☐ 2.6.9 (SNAPSHOT) ☐ 2.6.8

**Project Metadata**  
Group:   
Artifact:   
Name:   
Description:   
Package name:   
Packaging: ☒ Jar ☐ War  
Java ☐ 18 ☐ 17 ☒ 11 ☐ 8

**Dependencies**  
**Spring Web** WEB  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.  
**Thymeleaf** TEMPLATE ENGINES  
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.  
**Validation** I/O  
Bean Validation with Hibernate validator.

ADD DEPENDENCIES... CTRL + B

⇒ Generate

**Ha nem aktív a Run gomb várakozási idő után sem:**

File menu / Invalidate Chaches ...

**Indítsuk el a kiinduló alkalmazást => Működik.**

**pom.xml - dependencies**

spring-boot-starter-web, spring-boot-starter-test, spring-boot-starter-thymeleaf, **spring-boot-starter-validation**

Nézzük meg a **pom.xml** fájlt.

**Ha hibákat jelez: File menü / Invalidate Caches...**

**Néhány átnevezés (Refactor):**

Nevezzük át a **DemoApplication** osztályt **FoOsztaly** névre.

Nevezzük át a **com.example.demo** csomagot **urlap** névre.

Mivel a csomagot átneveztük, ezért meg kell adni hol van a main() metódust tartalmazó osztály, különben nem tudja futtatni azt:

Run menü / Edit configuration / Configuration / Main class:

urlap.FoOsztaly

Készítsünk egy **UzenetOsztaly** osztályt az **urlap** csomagba, ami fogadja az űrlap adatait, **getter** és **setter** metódusokkal.

Változói:

```
private long id;
private String content;
```

**package urlap;**

```
public class UzenetOsztaly {
 private long id;
 private String content;
 public long getId() {
```

```

 return id;
}
public void setId(long id) {
 this.id = id;
}
public String getContent() {
 return content;
}
public void setContent(String content) {
 this.content = content;
}
}

```

Itt más logikával dolgozza fel az űrlap adatait, mint a PHP, amit a Web-programozás tárgynál megismertek.

**A Controller-ben**, majd egy **modellben** továbbítunk egy **üres UzenetOszty** példányt a template-nek (nézet, HTML fájl). A nézetben majd ebbe az **UzenetOszty** példányba tesszük a kitöltött űrlap adatait és adjuk át egy **route**-nak, ami paraméterként továbbítja a hozzárendelt **metódusnak**.

Készítsünk egy kontrollert: **UrlapController.java**

```

package urlap;

import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class UrlapController {
 @GetMapping("/feladat")
 public String urlapForm(Model model) {
 model.addAttribute("attr1", new UzenetOszty());
 return "urlap";
 }
 @PostMapping("/feladat2")
 public String urlapSubmit(@ModelAttribute UzenetOszty uzenetOszty, Model model) {
 model.addAttribute("attr2", uzenetOszty);
 return "eredmeny";
 }
}

```

**A végén megnézzük**, hogy lehet megoldani 1 db URL-el.

Látjuk, hogy a fő osztályunkban, (ahol a main() van, FoOszty) nem kell hivatkozni a controller osztályunkra (UrlapController): **a Spring automatikusan megkeresi és felhasználja**. Lehet több controller osztályunk is, csak ne használjuk ugyanolyan URL-eket a Mappingoknál (@GetMapping, @PostMapping)

Az űrlap oldalának hívásához a GET módszert használjuk.

Ezért kell a **@GetMapping**. => betölti az urlap.html templatet (nézetet).

Ehhez használunk egy **modellt**, amiben továbbítjuk az "attr1" attribútummal a **UzenetOszta**ly példányt (new UzenetOszta

ly())  
Itt egy **üres UzenetOszta**ly példányt adunk át a **modellben** az "attr1" attribútummal az **urlap.html** nézetnek.

Az **urlap.html** nézetben majd ebbe az üres **UzenetOszta**ly példányba tesszük a kitöltött űrlap adatait és adjuk át a **@PostMapping("/feladat")** route-nak, ami paraméterként továbbítja a **urlapSubmit** metódusnak:

```
urlapSubmit(@ModelAttribute UzenetOszta
```

ly uzenetOszta

Mivel az űrlapnál majd **POST módszerrel** küldjük el az adatokat, ezért szükséges a **@PostMapping**.  
Ezt a részt az űrlap elküldése után nézzük meg.

**Készítsük el az urlap.html template-et (nézetet):**

Az Űrlapok kezelése és a HTML kód a Web-programozás tárgy anyaga, itt csak felhasználjuk.

```
src/main/resources/templates/urlap.html
<!DOCTYPE html>
<html lang="en">
<html xmlns:th="http://thymeleaf.org">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
 <h1>Form</h1>
 <form action="#" th:action="@{/feladat2}" th:object="${attr1}" method="post">
 <p>Id: <input type="text" th:field="*{id}" /></p>
 <p>Message: <input type="text" th:field="*{content}" /></p>
 <p><input type="submit" value="Submit" /> <input type="reset" value="Reset" /></p>
 </form>
</body>
</html>
```

A **th:action="@{/feladat2}"** kifejezés az űrlap adatait a /feladat2 route-ra irányítja POST módszerrel.

**A th:object="\${attr1}" kifejezés:**

A GreetingController-ben láttuk:

```
model.addAttribute("attr1", new UzenetOszta
```

```
ly());
```

átadtunk egy üres **UzenetOszta**ly példányt.

Itt ezt az üres **UzenetOszta**ly példányt töltjük ki az űrlap adataival

És majd a gombra való kattintás után ezt küldjük el a **@PostMapping("/feladat")** route-on keresztül a **greetingSubmit** metódusnak:

```
@PostMapping("/feladat2")
```

```
public String urlapSubmit(@ModelAttribute UzenetOszta
```

```
ly uzenetOszta
```

A két űrlap mező, amire a **th:field="{id}"** és **th:field="{content}"** kifejezésekkel hivatkozunk, megfelelnek a **UzenetOszta**ly osztály mezőinek.

A gombra kattintás után elküldjük az űrlap adatait a szervernek.

Feladat: az űrlap adatainak megjelenítése:

<http://localhost:8080/feladat2>

Result
id: 100
content: Hello, World!
<a href="#">Submit another message</a>



## Vissza a kontrollerre: UrlapController.java

```
.....
@PostMapping("/feladat2")
public String urlapSubmit(@ModelAttribute UzenetOsztaly uzenetOsztaly, Model model) {
 model.addAttribute("attr2", uzenetOsztaly);
 return "eredmeny";
}
```

Az urlapSubmit metódusnak két bemeneti paramétere van:

- **UzenetOsztaly uzenetOsztaly, Model model**

Az uzenetOsztaly objektumban érkeznek az űrlapnál elküldött adatok.

- **Model model:** A Model példányt az adatok továbbítására használjuk a nézetnek. A megkapott **UzenetOsztaly** példányt továbbítjuk benne: ebben vannak az űrlapról elküldött adatok.

**@ModelAttribute:** e nélkül is működik. Szerepe: ellenőrizni, hogy az űrlaptól kapott modellben van-e **UzenetOsztaly uzenetOsztaly**. Ha nincs, akkor tesz bele egy üres **UzenetOsztaly** példányt.

A továbbított adatokat feldolgozzuk az eredmény nézetben:

src/main/resources/templates/eredmeny.html

```
<!DOCTYPE html>
<html lang="en">
<html xmlns:th="http://thymeleaf.org">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
 <h1>Result</h1>
 <p th:text="id: ' + ${attr2.id} " />
 <p th:text="content: ' + ${attr2.content} " />
 Submit another message
</body>
</html>
```

Próbáljuk ki az alkalmazást!

<http://localhost:8080/feladat>

<http://localhost:8080/feladat2>

Az ID mezőbe egész számot adjunk meg.

A következő validációs feladatnál majd beállítjuk, hogy csak egész számot fogadjon el.

A feladatot egy db URL-el is meg tudjuk oldani, mert az egyik route GET, a másik POST:

**@GetMapping, @PostMapping**

## UrlapController.java

```
.....
@PostMapping("/feladat")
.....
urlap.html
.....
<form action="#" th:action="@{/feladat}" th:object="${attr1}" method="post">
.....
```

Írjuk át és így is próbáljuk ki!

## Feladat-2 - Űrlap adatainak szerver oldali validációja

Folytassuk az előző alkalmazást.

Állítsuk be a következő validációs szabályokat:

**Id:** nem lehet üres, csak egész szám lehet, legalább 2

**Message:** nem lehet üres, legalább 2 és legfeljebb 30

**Form**

Id:

nagyobbnak, vagy egyenlőnek kell lennie, mint 2

Message:

a méretnek a(z) 2 és 30 értékek között kell lennie

**Form**

Id:

Message:

a méretnek a(z) 2 és 30 értékek között kell lennie

**Form**

Id:

Egesz szamot adj meg!

Message:

**Result**

id: 345

content: szöveg2

[Submit another message](#)

### Megoldás

Pirossal jelezve a változtatásokat.

Írjuk be a UzenetOsztaly-ba a validációs szabályokat:

```
package urlap;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
public class UzenetOsztaly {
 @NotNull
 @Min(2)
 private long id;
 @NotNull
 @Size(min=2, max=30)
 private String content;
 public long getId() {
 return id;
 }
}
```

```

public void setId(long id) {
 this.id = id;
}
public String getContent() {
 return content;
}
public void setContent(String content) {
 this.content = content;
}
}

```

### Módosítsuk a kontrollert:

```

@Controller
public class UrlapController {
 @GetMapping("/feladat")
 public String urlapForm(Model model) {
// Az attributum neve uzenetOsztyal kell legyen, mert a @PostMapping részénél a return "urlap";-nál
// szintén az urlap-ot hívja meg hiba esetén és ott a uzenetOsztyal objektumban adja át az adatot.
 model.addAttribute("uzenetOsztyal", new UzenetOsztyal());
 return "urlap";
 }
 @PostMapping("/feladat2")
 public String urlapSubmit(@Valid @ModelAttribute UzenetOsztyal uzenetOsztyal, BindingResult
bindingResult, Model model) {
 if (bindingResult.hasErrors())
 return "urlap";
 model.addAttribute("attr2", uzenetOsztyal);
 return "eredmeny";
 }
}

```

### Módosítsuk az urlap.html-t:

```

.....
<form action="#" th:action="@{/feladat2}" th:object="${uzenetOsztyal}" method="post">
 <p>Id: <input type="text" th:field="*{id}" /></p>
 <p th:if="${#fields.hasErrors('id')}" th:errors="*{id}"></p>
 <p>Message: <input type="text" th:field="*{content}" /></p>
 <p th:if="${#fields.hasErrors('content')}" th:errors="*{content}"></p>
 <p><input type="submit" value="Submit" /> <input type="reset" value="Reset" /></p>
</form>
.....

```

Ha ez így maradna, akkor is jó lenne a megoldás, de nagyon hosszú validációs hibaüzenetet írna ki:

Form

Id:

Failed to convert property value of type java.lang.String to required type int for property id; nested exception is java.lang.NumberFormatException: For input string: "szöveg1"

Message:

### Készítsünk egyéni validációs hibaüzenetet:

src\main\resources\messages.properties fájlt készíteni

File menü / New / File

és beírni:

**typeMismatch.uzenetOsztaly.id=Egesz szamot adj meg!**

Az uzenetOsztaly class-ban lévő id változó, ami int típusú, nem int értéket kapna az űrlapon,  
akkor ezt írja ki: Egesz szamot adj meg!

vagy általánosan:

**typeMismatch.int=Egesz szamot adj meg!**

A többi üzenet is megváltoztatható:

pl. ehhez a szabályhoz:

@Min(2)

javax.validation.constraints.Min.message = Legyen nagyobb vagy egyenlo, mint {value}

**Sok egyéb beállítási lehetőség:**

<https://docs.oracle.com/javaee/7/tutorial/bean-validation001.htm>

<https://terasolunaorg.github.io/guideline/5.0.2.RELEASE/en/ArchitectureInDetail/Validation.html>

src\main\resources\messages.properties fájlt készíteni:

**typeMismatch.uzenetOsztaly.id=Egesz szamot adj meg!**

vagy általánosan:

**typeMismatch.int=Egesz szamot adj meg!**

Az eredeti üzenet is megváltoztatható:

pl. ehhez a szabályhoz:

@Min(2)

javax.validation.constraints.Min.message = Legyen nagyobb vagy egyenlo, mint {value}

<https://terasolunaorg.github.io/guideline/5.0.2.RELEASE/en/ArchitectureInDetail/Validation.html>

## 7-8. gyakorlat

A: Spring-Boot-Adatbázis-JPA - Hibernate-MySQL-CRUD alkalmazás

**CRUD:** Create, Read, Update, Delete

<https://www.codejava.net/frameworks/spring-boot/spring-boot-crud-example-with-spring-mvc-spring-data-jpa-thymeleaf-hibernate-mysql>

<https://www.geeksforgeeks.org/spring-mvc-crud-with-example/>

<https://www.javaguides.net/2020/05/spring-boot-crud-web-application-with-thymeleaf.html>

<https://javatechonline.com/spring-boot-mvc-crud-example/>

**JPA:** Persist data in SQL stores with **Java Persistence API** using Spring Data and **Hibernate**.

<https://spring.io/projects/spring-data-jpa>

**persistent** = állandó, tartós

**persist:** a memóriában lévő adatot kiírjuk háttértárra, adatbázisba.

A memóriában lévő adat azért nem tartós, mert pl. a gép kikapcsolásával elveszik.

**Hibernate:** A Hibernate egy **objektum-relációs leképezést (ORM)** megvalósító programkönyvtár Java platformra. Segítségével osztályokat és a relációs adatbázisok tábláit tudjuk egymásba leképezni, az adatbázisban lévő rekordokat objektumokként kezelhetjük, és az objektumainkat egyszerűen tárolhatjuk állapotmegőrző módon adattáblákban.

**A JDBC-hez képest sokkal egyszerűbben tudjuk elérni az adatbázist.**

<https://hibernate.org/>

**Objektum-relációs leképezés (ORM) sok nyelvben, keretrendszerben van. pl.:**

Laravel: <https://laravel.com/docs/9.x/eloquent>

C#/NET: <https://education.launchcode.org/csharp-web-development/chapters/orm-intro/background.html>

Node.js: <https://www.section.io/engineering-education/introduction-to-sequalize-orm-for-nodejs/>

Python: <https://www.fullstackpython.com/object-relational-mappers-orms.html>

### Feladat

Készítsünk egy CRUD alkalmazást (Create, Read, Update, Delete) dolgozók adatainak kezeléséhez.

Create: Tudjunk új dolgozót felvinni

Read: jelenítsük meg a dolgozók adatait a kezdőoldalon

Update: Tudjuk egy adott dolgozó adatait módosítani

Delete: Tudjunk egy adott dolgozót törölni

### Főoldal

<http://localhost:8888/>

Dolgozók					
<a href="#">Új dolgozót hozzáad</a>					
ID	Név	Cím	Kor		
1	Tóth István	Debrecen	35	<a href="#">Edit</a>	<a href="#">Delete</a>
2	Nagy Éva	Szeged	20	<a href="#">Edit</a>	<a href="#">Delete</a>
3	Horváth Péter	Kecskemét	23	<a href="#">Edit</a>	<a href="#">Delete</a>

**Új dolgozó hozzáadása**

<http://localhost:8888/uj>

**Hozzáadás után**

<http://localhost:8888/>

## Új dolgozó hozzáadása

Új dolgozó hozzá lett adva! ID=4

## Dolgozók

[Új dolgozót hozzáad](#)

ID	Név	Cím	Kor		
1	Tóth István	Debrecen	35	<a href="#">Edit</a>	<a href="#">Delete</a>
2	Nagy Éva	Szeged	20	<a href="#">Edit</a>	<a href="#">Delete</a>
3	Horváth Péter	Kecskemét	23	<a href="#">Edit</a>	<a href="#">Delete</a>
4	Kiss Judit	Szolnok	37	<a href="#">Edit</a>	<a href="#">Delete</a>

Ha olyat akarunk hozzáadni akinek  
a neve és címe már szerepel  
a rendszerben

<http://localhost:8888/uj>

## Új dolgozó hozzáadása

Ezzel a névvel és címmel már van dolgozó. ID=2

## Dolgozók

[Új dolgozót hozzáad](#)

ID	Név	Cím	Kor		
1	Tóth István	Debrecen	35	<a href="#">Edit</a>	<a href="#">Delete</a>
2	Nagy Éva	Szeged	22	<a href="#">Edit</a>	<a href="#">Delete</a>
3	Horváth Péter	Kecskemét	23	<a href="#">Edit</a>	<a href="#">Delete</a>
4	Kiss Judit	Szolnok	37	<a href="#">Edit</a>	<a href="#">Delete</a>

## Módosítás

<http://localhost:8888/edit/2>

## Dolgozó módosítása

## Módosítás után

<http://localhost:8888/>

Dolgozó módosítva! ID=2

## Dolgozók

[Új dolgozót hozzáad](#)

ID	Név	Cím	Kor		
1	Tóth István	Debrecen	35	<a href="#">Edit</a>	<a href="#">Delete</a>
2	Nagy Éva	Szeged	22	<a href="#">Edit</a>	<a href="#">Delete</a>
3	Horváth Péter	Kecskemét	23	<a href="#">Edit</a>	<a href="#">Delete</a>
4	Kiss Judit	Szolnok	37	<a href="#">Edit</a>	<a href="#">Delete</a>

## Törlés után

<http://localhost:8888/>

Dolgozó törölve! ID=3

## Dolgozók

[Új dolgozót hozzáad](#)

ID	Név	Cím	Kor		
1	Tóth István	Debrecen	35	<a href="#">Edit</a>	<a href="#">Delete</a>
2	Nagy Éva	Szeged	22	<a href="#">Edit</a>	<a href="#">Delete</a>
4	Kiss Judit	Szolnok	37	<a href="#">Edit</a>	<a href="#">Delete</a>

### Megoldás

Az egyszerűség miatt itt csak @GetMapping és @PostMapping útvonalakat használunk, lehetne használni még @PutMapping és @DeleteMapping útvonalakat is. Azokra nézünk majd példát a RESTful fejezetben.

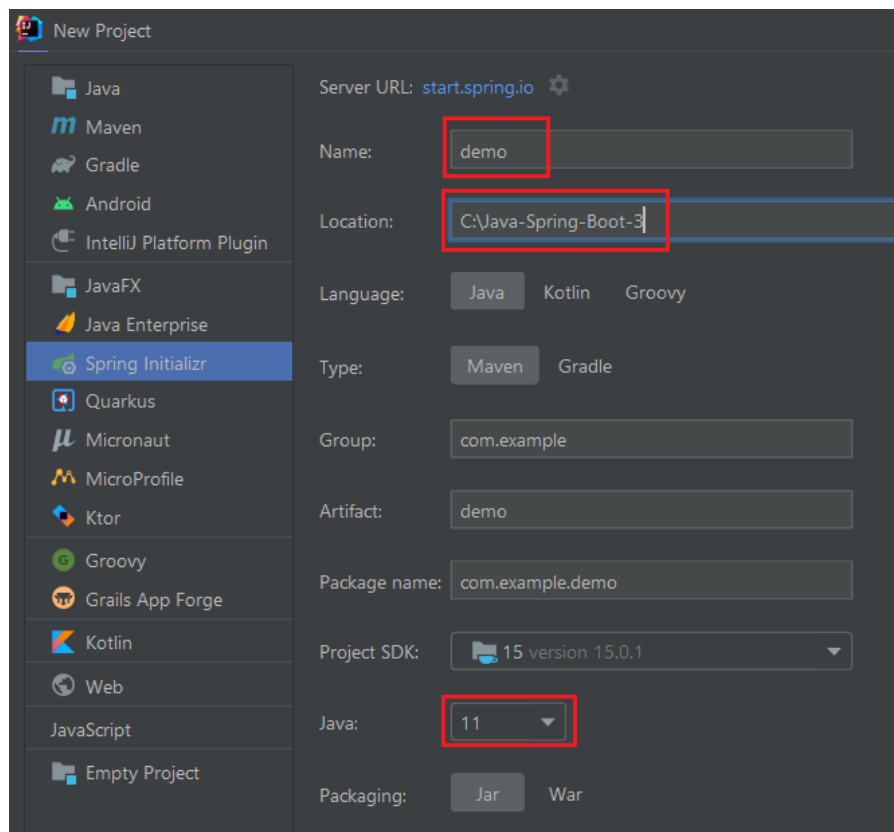
Készítsünk egy mappát a projektnek pl. **c:\Java-Spring-Boot-3**

### Előkészítés

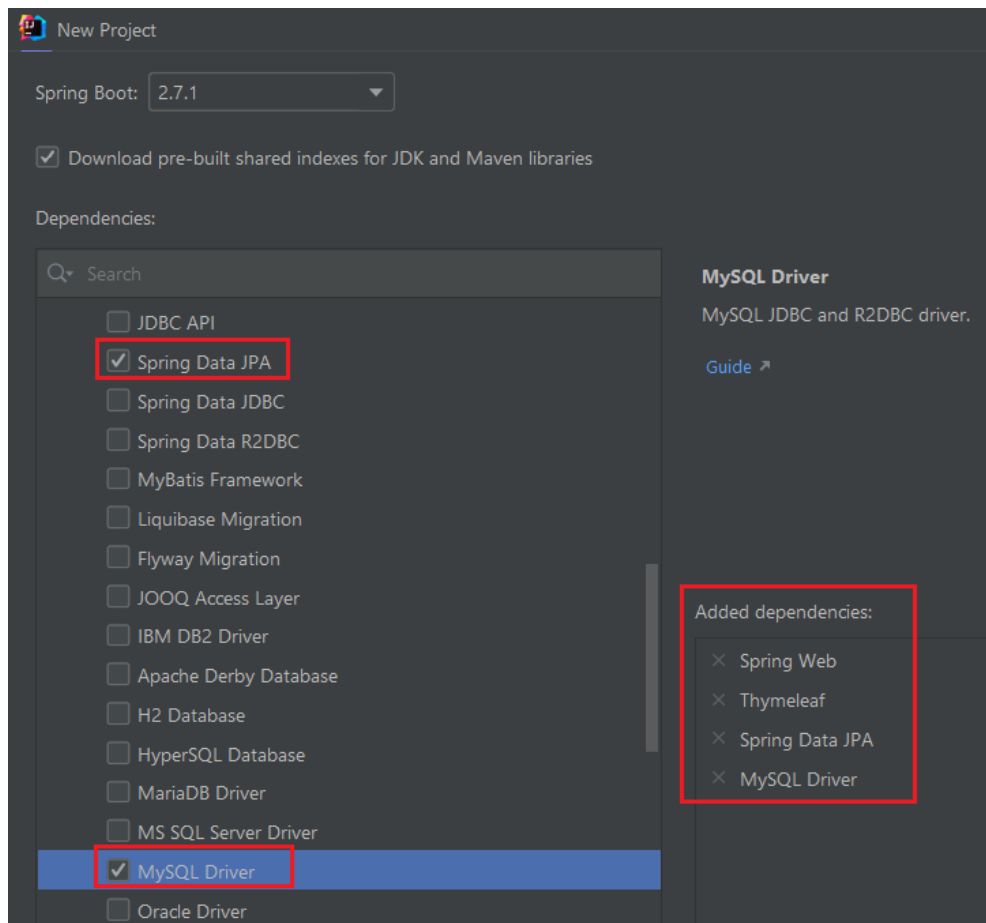
Töltsük le az alaprendszert. Források.zip-be is bemásoltam.

**A, módszer**

**IntelliJ / New project / Spring Initializr /**



Next



**Finish**

vagy:

**B, módszer: volt amikor hibát adott!**

<https://start.spring.io/>

=> Generate

Nézzük meg a **pom.xml** fájlt.



Ha hibákat jelez: File menü / Invalidate Caches...

**Ha nem aktív a Run gomb várakozási idő után sem:**

File menu / Invalidate Chaches ...

**Ha most indítjuk el a kiinduló alkalmazást még nem működik!**

Hibaüzenet: Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.

**A következők is kellenek a helyes működéshez:**

**XAMPP, MySQL, phpMyAdmin indítása**

Már van ez az üres fájl a projektben: src/main/resources/**application.properties**. Töltsük fel a következő tartalommal. Ebben adjuk meg az adatbázishoz való csatlakozás paramétereit:

**server.port=8888**

spring.datasource.url=jdbc:mysql://\${MYSQL\_HOST:**localhost**}:3306/**feladat**

spring.datasource.username=**root**

spring.datasource.password=

spring.datasource.driver-class-name =com.mysql.**jdbc.Driver**

spring.jpa.hibernate.ddl-auto=update

**Indítsuk el az alkalmazást => Működik.**

Importálják be a **dolgozo.sql** adatbázisát a **Források.zip** fájlból.

Nézzük meg a beimportált adatbázis **dolgozo** tábláját.

Nézzük meg a **pom.xml** fájlt, hogy milyen dependency-ket használ.

spring-boot-starter-web, spring-boot-starter-test, spring-boot-starter-thymeleaf

**spring-boot-starter-data-jpa, mysql-connector-java**

**Néhány átnevezés (Refactor):**

- package com.example.demo;	=> com.example. <b>crud</b> ;
- class DemoApplication	=> class <b>CrudApplication</b>

**Készítsük el a Dolgozo @Entity Modelt (Java osztály):**

A Model osztály segítségével fogunk kapcsolatot tartani az adatbázisban.

Mezők: id, nev, cim, kor

src/main/java/com/example/crud -on állva: File menü / New / Java Class

src/main/java/com/example/**crud/Dolgozo.java**

**Vegyük fel az az osztályba a változókat, amelyek szükségesek az adatbázissal való kapcsolathoz.**

```
package com.example.crud;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
```

**@Entity**

```
public class Dolgozo {
```

```
 @Id
```

```

@GeneratedValue(strategy=GenerationType.IDENTITY) // AUTO_INCREMENT
private int id;
private String nev;
private String cim;
private int kor;
}

```

Ez a legegyszerűbb eser, ahol az osztály neve megegyezik az adatbázis táblanévvel és a változónevek megegyeznek a mezőnevekkel a táblában. Ha ezek eltérőek, akkor a következő annotációkat kell használni:

```

@Entity
@Table(name = "tablanev")
public class Dolgozo {
 @Id
 @GeneratedValue(strategy=GenerationType.IDENTITY) // AUTO_INCREMENT
 private int id;
 @Column(name = "dolgozo_nev")
 private String nev;
 private String cim;
 @Column(name = "eletkor")
 private int kor;
}

```

**IntelliJ-vel generáltassuk le a Getter és Setter metódusokat az osztály végéhez.**

**Code menü / Generate / Getter and Setter => mezők kijelölése**

**Készítsük el a DolgRepo interfészt (Dolgozó Repository):**

**Repository:** majd látjuk: .... extends CrudRepository....

**Repository:** Jelentősen leegyszerűsítik az adatbázishasználatot

<https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html>

Az adatbázis műveletek ORM (Object Relational Mapping) megvalósításánál a gyakori CRUD (Create, Read, Update, Delete) műveletek alkalmazásához hozták létre a Spring JPA-ban a Repository-kat.

Ezek interfészek, amiket csak fel kell használni.

### **CrudRepository**

A CRUD műveletekhez (Create, Read, Update, Delete) készen vannak már a Repository interfészek a Spring JPA-ban és ki van dolgozva a háttérben, hogy a rendszer hogyan segítse az ezeket implementáló osztályokat pl. az adatbázisműveletekhez.

Előnye: használatával CRUD műveleteknél tovább csökkentjük a felesleges kódok írását

<https://docs.spring.io/spring-data/data-commons/docs/1.6.1.RELEASE/reference/html/repositories.html>

**A CrudRepository interface fontosabb metódusait:**

`count()`, `delete(T entity)`, `deleteAll()`, `deleteById(ID id)`, `existsById(ID id)`,  
`findAll()`, `findById(ID id)`, `save(S entity)`, `saveAll(Iterable<S> entities)`

<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html>

**A Repository fő interfésze a Repository interface:**

<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/Repository.html>

Itt megtaláljuk az összes már elkészített al-intefészt is. pl. CrudRepository<T,ID>  
Az oldalon látjuk:

Interface Repository<T,ID>

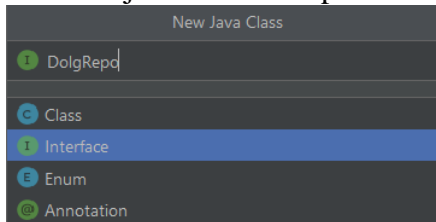
T - the domain type the repository manages

ID - the type of the id of the entity the repository manages

Például majd a példában: CrudRepository<User, Integer>

**Mivel interfészt használunk, ezért kell majd egy osztály készíteni, ami implementálja ezt az interfészt, vagy a Dependency injection is megoldja ezt (lásd később)**

src/main/java/com/example/adatbazis -on állva: File menü / New / Java Class



src/main/java/com/example/crud/**DolgRepo.java**

```
package com.example.crud;
import org.springframework.data.repository.CrudRepository;
public interface DolgRepo extends CrudRepository<Dolgozo, Integer> {
}
```

**Csak ennyit kell megadnunk és a Java Spring a háttében biztosítja az egyszerű adatbázisműveleteket!**

**Készítsük el a kontroller osztályt:**

src/main/java/com/example/crud/**Vezerlo.java**

src/main/java/com/example/crud -on állva: File menü / New / Java Class

A kontrollerben felhasználjuk a **Dependency Injection**-t.

lásd a következő MainController osztálynál.

**@Autowired**

**private DolgRepo dolgRepo;**

<https://www.baeldung.com/spring-dependency-injection>

<https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring>

<https://www.javatpoint.com/dependency-injection-in-spring>

**Injektálás:** nem mi hoztunk implementáljuk a DolgRepo interfészt, hanem a rendszer implementálja és beinjektálja az osztályba.

**Más nyelven is keretrendszerekben is van Dependency Injection. pl.**

Laravel: <https://laravel.com/docs/9.x/container>

C#/.NET: <https://docs.microsoft.com/en-us/dotnet/core/extensions/dependency-injection>

Node.js: <https://blog.risingstack.com/dependency-injection-in-node-js/>

Python: <https://python-dependency-injector.ets-labs.org/>

src/main/java/com/example/crud/**Vezerlo.java**

```
package com.example.crud;
```

```

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.servlet.mvc.support.RedirectAttributes;

```

**@Controller**

```
public class Vezerlo {
```

```
 @Autowired
```

```
 private DolgRepo dolgRepo;
```

```
// Főoldal
```

```
 @GetMapping("/")
```

```
 public String Fooldal(Model model, String uzenet) {
 model.addAttribute("dolgozok", dolgRepo.findAll());
 model.addAttribute("uzenet", model.getAttribute("uzenet"));
 return "index";
 }

```

```
// Új dolgozó hozzáadása oldal meghívása
```

```
 @GetMapping("/uj")
```

```
 public String UjDolgozoOldal(Model model) {
 model.addAttribute("dolgozo", new Dolgozo());
 return "ujdolgozo";
 }

```

```
// Új dolgozó hozzáadása: mentés az adatbázisba
```

```
// RedirectAttributes: a return "redirect:/" ; -nél ezzel tudunk adatot átvinni (Model-el nem lehet)
```

```
// Ít egy üzenetet viszünk át: redirAttr.addFlashAttribute("uzenet", ".....
```

```
 @PostMapping(value = "/ment")
```

```
 public String mentDolgozo(@ModelAttribute Dolgozo dolgozo, RedirectAttributes redirAttr) {
```

```
// Megnézzük, hogy van-e már dolgozó ezzel a névvel és címmel
```

```
 for(Dolgozo dolgozo2: dolgRepo.findAll())
```

```
 if(dolgozo2.getNev().equals(dolgozo.getNev()) &&
```

```
 dolgozo2.getCim().equals(dolgozo.getCim())){
```

```
// Ha már van ilyen dolgozó, akkor nem visszük fel az adatbázisba:
```

```
 redirAttr.addFlashAttribute("uzenet", "Ezzel a névvel és címmel már van dolgozó.
```

```
 ID="+dolgozo2.getId());
```

```
// Vissza a főoldalra:
```

```
 return "redirect:/";
```

```
 }
```

```
// Ha nincs ilyen dolgozó, akkor felvisszük az adatbázisba:
```

```
 dolgRepo.save(dolgozo);
```

```
 redirAttr.addFlashAttribute("uzenet", "Új dolgozó hozzá lett adva! ID="+dolgozo.getId());
```

```
 return "redirect:/";
```

```
 }
```

```
// Az adott ID-jű dolgozó módosítása oldal meghívása
```

```
 @GetMapping("/edit/{id}")
```

```
 public String modositDolgozo(@PathVariable(name = "id") int id, Model model) {
```

```

 model.addAttribute("dolgozo", dolgRepo.findById(id));
 return "modosit";
 }

// Az adott ID-jű dolgozó módosítása: rekord módosítása az adatbázisban
// A CrudRepository-nak nincs Update metódusa:
// https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html
// Az Update megvalósítása: Save metódussal: dolgRepo.save(dolgozo);
// Ha a dolgozó példányban az ID megegyezik a módosítandó rekor ID-jével,
// akkor Update (az adott rekord módosítása), különben Insert (Új rekord felvitele)
// Ezért kell majd a modosit.html -ben:
// <p><input type="hidden" th:field="*{id}" value= "${dolgozo.id}" /></p>
 @PostMapping(value = "/modosit")
 public String modositDolgozo(@ModelAttribute Dolgozo dolgozo, RedirectAttributes redirAttr) {
 dolgRepo.save(dolgozo);
 redirAttr.addFlashAttribute("uzenet", "Dolgozó módosítva! ID="+dolgozo.getId());
 return "redirect:/";
 }

// Az adott ID-jű dolgozó törlése
 @GetMapping("/delete/{id}")
 public String torolDolgozo(@PathVariable(name = "id") int id, RedirectAttributes redirAttr) {
 redirAttr.addFlashAttribute("uzenet", "Dolgozó törölve! ID="+dolgRepo.findById(id).get().getId());
 dolgRepo.delete(dolgRepo.findById(id).get());
 return "redirect:/";
 }
}

```

### Készítsük el a nézeteket.

src\main\resources\templates mappában:  
File menü / New / HTML file

```

src\main\resources\templates\index.html
<!DOCTYPE html>
<html lang="en">
<html xmlns:th="https://www.thymeleaf.org">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
<div>
 <h3 style="color:red;" th:text="${uzenet}"></h3>
 <h1>Dolgozók</h1>
 Új dolgozót hozzáad

 <table>
 <thead>
 <tr>
 <th>ID</th>
 <th>Név</th>
 <th>Cím</th>

```



```
<p><input type="text" th:field="*{kor}" value= "${dolgozo.kor}" required/></p>
<p><input type="submit" value="Módosít" /></p>
</form>
</body>
</html>
```

## B: Többszámlás feladat

<https://www.geeksforgeeks.org/how-to-implement-one-to-many-mapping-in-spring-boot/>  
<https://hellokoding.com/jpa-one-to-many-relationship-mapping-example-with-spring-boot-hsql/>  
<https://attacomsian.com/blog/spring-data-jpa-one-to-many-mapping>  
<https://www.stackchief.com/blog/One%20To%20Many%20Example%20%7C%20Spring%20Data%20JPA>  
<https://thorben-janssen.com/best-practices-many-one-one-many-associations-mappings/>  
<https://www.callicoder.com/hibernate-spring-boot-jpa-one-to-many-mapping-example/>  
<https://codebun.com/spring-data-jpa-one-to-many-mapping-example/>  
<https://www.javaguides.net/2019/08/spring-boot-jpa-hibernate-one-to-many-example-tutorial.html>  
<https://www.appsdeveloperblog.com/one-to-many-mapping-hibernate-jpa-using-spring-boot-and-mysql/>  
<https://medium.com/huawei-developers/database-relationships-in-spring-data-jpa-8d7181f50f60>  
<https://asbnotebook.com/jpa-one-to-many-example-spring-boot/>  
<https://howtodoinjava.com/hibernate/hibernate-one-to-many-mapping/>  
<https://www.bezkoder.com/jpa-one-to-many/>

### Feladat

#### 3 tábla

szemely(**id**, nev, *rendszam*, magassag)

telefon(**id**, *szemelyid*, szam)

jarmu(**rendszm**, marka, szín)

#### 3 osztály a táblákkal való kapcsolathoz

##### Személy

id: int

név: String

rendszám: String

magasság: int

##### Telefon

id: int

személyid: int

szám: String

##### Jármű

rendszm: String

márka: String

szín: String

**A személy és a telefon táblák között 1:N (egy a többhöz) kapcsolat van:** egy személynek lehet több telefonszáma is, de egy telefonszám csak egy személyhez tartozhat.

**A személy és a jármű táblák között 1:1 (egy az egyhez) kapcsolat van:** egy személyhez csak egy jármű tartozhat és egy járműhöz csak egy személy tartozhat.

Importáljuk be az adatokat a **szemelyek.sql** fájlból az adatbázisba. Nézzük meg a táblákat.

**Jelenítsük meg a főoldalon a következő feladatok eredményét:**

**A, Írassa ki a táblák adatait a következő formában:**

<http://localhost:1000/>



1; Tóth Ferenc; ABC123; 175  
2; Kiss József; FGH456; 168  
3; Horváth Mária; SDF345; 165  
4; Kerekes Katalin; HJK678; 160

1; 1; 345678  
2; 2; 456123  
3; 2; 234678  
4; 3; 345123  
5; 3; 123789  
6; 3; 345987

ABC123; Ford; piros  
FGH456; Skoda; kék  
HJK678; BMW; piros  
SDF345; Ford; fehér

**B, Írassa ki minden személy minden adatát a következő formában:**

<http://localhost:1000/szemelyek>

1; Tóth Ferenc; ABC123; 175; 345678; Ford; piros;  
2; Kiss József; FGH456; 168; 456123, 234678; Skoda; kék;  
3; Horváth Mária; SDF345; 165; 345123, 123789, 345987; Ford; fehér;  
4; Kerekes Katalin; HJK678; 160; ; BMW; piros;

Az egyes elemeket (;) karakterrel válasszuk el, a telefonszámokat (,) karakterrel

**C, Írassunk ki minden telefonszámot, mellé a tulajdonos nevét és járművének színét a következő formában:**

<http://localhost:1000/telefonok>

345678; Tóth Ferenc; piros  
456123; Kiss József; kék  
234678; Kiss József; kék  
345123; Horváth Mária; fehér  
123789; Horváth Mária; fehér  
345987; Horváth Mária; fehér

**D, Írassuk ki Kiss József nevét, magasságát és járművének típusát:**

<http://localhost:1000/adottszemely>

Kiss József; 168; Skoda

### Megoldás

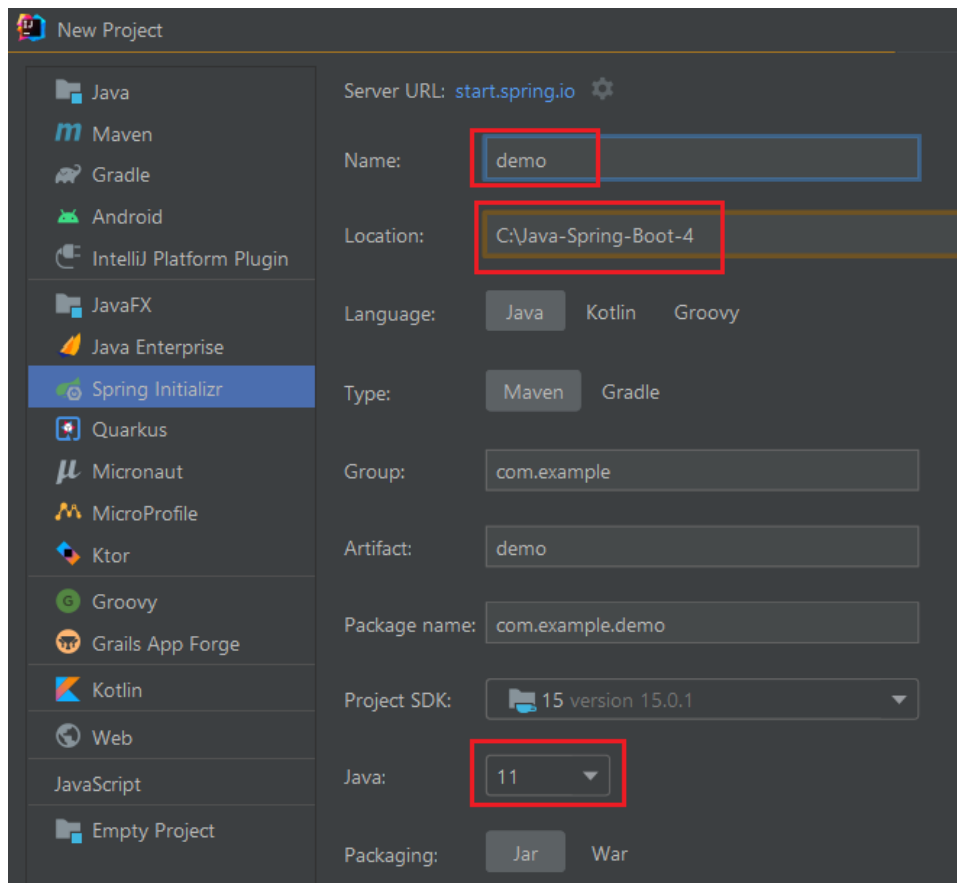
Készítsünk egy mappát a projektnek pl. c:\Java-Spring-Boot-4

### Előkészítés

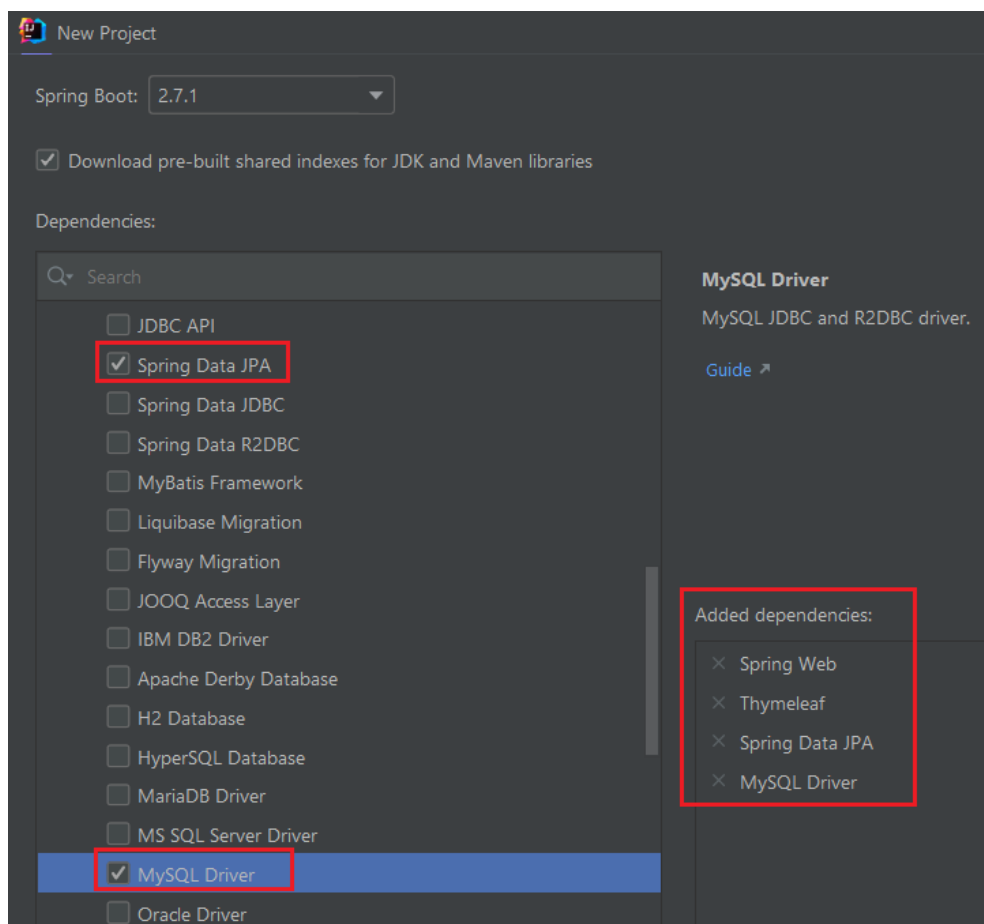
Az előző alkalmazás alaprendszerét használjuk itt is. Források.zip-ből másolható.

### **A módszer**

IntelliJ / New project / Spring Initializr /



Next



Finish

vagy:

**B, módszer: volt amikor hibát adott**

<https://start.spring.io/>

The screenshot shows the Spring Initializr web form. Red boxes highlight the following elements:

- Project**: Maven Project (selected)
- Language**: Java (selected)
- Spring Boot**: 2.7.0 (selected)
- Project Metadata**: Group (com.example), Artifact (demo), Name (demo), Description (Demo project for Spring Boot), Package name (com.example.demo), Packaging (Jar)
- Dependencies**: Spring Web (WEB), Thymeleaf (TEMPLATE ENGINES), Spring Data JPA (SQL), MySQL Driver (SQL)
- ADD DEPENDENCIES... CTRL + B** button
- Java** version: 11 (selected)

=> Generate

Nézzük meg a **pom.xml** fájlt.

**Ha hibákat jelez: File menü / Invalidate Caches...**

src/main/resources/**application.properties**.

**server.port=1000**

spring.datasource.url=jdbc:mysql://\${MYSQL\_HOST:localhost}:3306/feladat

spring.datasource.username=**root**

spring.datasource.password=

spring.datasource.driver-class-name =com.mysql.jdbc.**Driver**

spring.jpa.hibernate.ddl-auto=update

**Ha nem aktív a Run gomb várakozási idő után sem:**

File menu / Invalidate Chaches ...

**Indítsuk el az alkalmazást => Működik.**

1, Táblák közötti kapcsolat nélkül (Fooldal részhez)

**Készítsük el a Személy @Entity Modelt (Java osztály):**

File menü / New / Java Class

src/main/java/com/example/demo/**Személy.java**

```
package com.example.demo;
```

```
import javax.persistence.Column;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.Id;
import javax.persistence.Table;
```

```
@Entity
@Table(name = "szemely")
public class Személy {
 @Id
 private int id;
 @Column(name = "nev")
 private String név;
 @Column(name = "rendszám")
 private String rendszám;
 @Column(name = "magassag")
 private int magasság;
}
```

**IntelliJ-vel generáltassuk le a Getter és Setter metódusokat az osztály végéhez.**  
**Code menü / Generate / Getter and Setter => mezők kijelölése**

**Készítsük el a Telefon @Entity Modelt:**

File menü / New / Java Class

src/main/java/com/example/demo/**Telefon.java**

```
package com.example.demo;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
```

```
@Entity
public class Telefon {
 @Id
 private int id;
 @Column(name = "szemelyid")
 private int személyid;
 @Column(name = "szam")
 private String szám;
}
```

**IntelliJ-vel generáltassuk le a Getter és Setter metódusokat az osztály végéhez.**  
**Code menü / Generate / Getter and Setter => mezők kijelölése**

**Készítsük el a Jármű @Entity Modelt:**

File menü / New / Java Class

src/main/java/com/example/demo/**Jármű.java**

```
package com.example.demo;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
```

```
@Entity
@Table(name = "jarmu")
```

```
public class Jármű {
 @Id
 private String rndszm;
 @Column(name = "marka")
 private String márka;
 @Column(name = "szin")
 private String szín;
}
```

**IntelliJ-vel generáltassuk le a Getter és Setter metódusokat az osztály végéhez.**  
**Code menü / Generate / Getter and Setter => mezők kijelölése**

**Készítsük el a SzemélyRepo interfészt (Személy Repository):**

File menü / New / Java Class => Interface

```
src/main/java/com/example/demo/SzemélyRepo.java
package com.example.demo;
import org.springframework.data.repository.CrudRepository;
public interface SzemélyRepo extends CrudRepository<Személy, Integer>{
}
```

**Készítsük el a TelefonRepo interfészt (Telefon Repository):**

File menü / New / Java Class => Interface

```
src/main/java/com/example/demo/TelefonRepo.java

package com.example.demo;
import org.springframework.data.repository.CrudRepository;
public interface TelefonRepo extends CrudRepository<Telefon, Integer>{
}
```

**Készítsük el a JárműRepo interfészt (Jármű Repository):**

File menü / New / Java Class => Interface

```
src/main/java/com/example/demo/JárműRepo.java
package com.example.demo;
import org.springframework.data.repository.CrudRepository;
public interface JárműRepo extends CrudRepository<Jármű, Integer>{
}
```

**Készítsük el a kontroller osztályt:**

src/main/java/com/example/demo/**Vezerlo.java**

File menü / New / Java Class

```
package com.example.demo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;

@Controller
```

```

public class Vezerlo {
 @Autowired
 private SzemélyRepo személyRepo;
 @Autowired
 private TelefonRepo telefonRepo;
 @Autowired
 private JárműRepo járműRepo;

 @GetMapping("/")
 public String Fooldal(Model model, String uzenet) {
 String str = A();
 model.addAttribute("str", str);
 return "index";
 }

 String A(){
 String str="";
 for(Személy személy: személyRepo.findAll()){
 str+=személy.getId()+" ";
 str+=személy.getNév()+" ";
 str+=személy.getRendszám()+" ";
 str+=személy.getMagasság();
 str+="
";
 }
 str+="
";
 for(Telefon telefon: telefonRepo.findAll()){
 str+=telefon.getId()+" ";
 str+=telefon.getSzemélyid()+" ";
 str+=telefon.getSzám();
 str+="
";
 }
 str+="
";
 for(Jármű jármű: járműRepo.findAll()){
 str+=jármű.getRndszm()+" ";
 str+=jármű.getMárka()+" ";
 str+=jármű.getSzín();
 str+="
";
 }
 return str;
 }
}

```

### Készítsük el a nézetet.

src\main\resources\templates mappában:

File menü / New / HTML file

src\main\resources\templates\index.html

```

<!DOCTYPE html>
<html lang="en">
<html xmlns:th="https://www.thymeleaf.org">
<head>
 <meta charset="UTF-8">
 <title>Title</title>
</head>
<body>
 <p th:utext="${str}"></p>
</body>
</html>

```

## Futtassuk az alkalmazást!

<http://localhost:1000/>

## 2, Táblák közötti kapcsolattal

### Magyarázat:

@JoinColumn(name = "rendszam", referencedColumnName = "randszm", insertable=false, updatable=false)

A insertable=false, updatable=false kell, ha már van rendszam mező a táblában és azon nem akarok változtatni. Különböző próbál létrehozni egy mezőt rendszam néven (hibaüzenet: nem tud, mert már van ilyen)

### @JoinColumn... és mappedBy ... közötti különbség

A @JoinColumn... -t annál a táblánál használjuk, amelyikben az idegen kulcs van.

A mappedBy ... -t NEM annál a táblánál használjuk, amelyikben az idegen kulcs van.

Az előző osztályokat kell kiegészíteni. Pirossal jelölve a változtatásokat:

```
.....
public class Személy {
```

```
 @Id
```

```
 private int id;
```

```
 @Column(name = "nev")
```

```
 private String név;
```

```
 @Column(name = "rendszam")
```

```
 private String rendszám;
```

```
 @Column(name = "magassag")
```

```
 private int magasság;
```

```
 @OneToMany(mappedBy = "személyid")
```

```
 private List<Telefon> telefonszámok;
```

```
 @OneToOne
```

```
 @JoinColumn(name = "rendszam", referencedColumnName = "randszm", insertable=false,
updatable=false)
```

```
 private Jármű jármű;
```

```
.....
getter és setter metódusok
```

```
.....
public class Telefon {
```

```
 @Id
```

```
 private int id;
```

```
 @Column(name = "szemelyid")
```

```
 private int személyid;
```

```
 @Column(name = "szam")
```

```
 private String szám;
```

```
 @ManyToOne
```

```
 @JoinColumn(name = "szemelyid", referencedColumnName = "id", insertable=false, updatable=false)
```

```
 private Személy személy;
```

```
.....
getter és setter metódusok
```

```

public interface SzemélyRepo extends CrudRepository<Személy, Integer>{
 Személy findByNév(String név);
}

public class Vezerlo {
 @Autowired
 private SzemélyRepo személyRepo;
 @Autowired
 private TelefonRepo telefonRepo;
 @Autowired
 private JárműRepo járműRepo;

 @GetMapping("/")
 public String Fooldal(Model model, String uzenet) {
 String str = A();
 model.addAttribute("str", str);
 return "index";
 }

 @GetMapping("/szemelyek")
 public String SzemélyekAdatai(Model model, String uzenet) {
 String str = B();
 model.addAttribute("str", str);
 return "index";
 }

 @GetMapping("/telefonok")
 public String TelefonokAdatai(Model model, String uzenet) {
 String str = C();
 model.addAttribute("str", str);
 return "index";
 }

 @GetMapping("/adottszemely")
 public String AdottSzemélyAdatai(Model model, String uzenet) {
 String str = D();
 model.addAttribute("str", str);
 return "index";
 }

 String A(){
 String str="";
 for(Személy személy: személyRepo.findAll()){
 str+=személy.getId()+" "+személy.getNév()+" "+személy.getRendszám()+" "+személy.getMagasság();
 str+="
";
 }
 str+="
";
 for(Telefon telefon: telefonRepo.findAll()){
 str+=telefon.getId()+" "+telefon.getSzemélyid()+" "+telefon.getSzám();
 str+="
";
 }
 }
}

```



```

 str+="
";
 for(Jármű jármű: járműRepo.findAll()){
 str+=jármű.getRndszm()+" "; "+jármű.getMárka()+" "; "+jármű.getSzín();
 str+="
";
 }
 return str;
}

String B(){
 String str="";
 for(Személy személy: személyRepo.findAll()){
 str+=személy.getId()+" "; "+személy.getNév()+" "; "+személy.getRendszám()+" ";
 "+személy.getMagasság()+" ";
 for(int i=0;i<személy.getTelefonszámok().size();i++){
 str+=személy.getTelefonszámok().get(i).getSzám();
 if(i<személy.getTelefonszámok().size()-1)
 str+=", ";
 }
 str+=" ";
 str+=személy.getJarmű().getMárka()+" "; "+személy.getJarmű().getSzín()+" ";
 str+="
";
 }
 return str;
}

String C(){
 String str="";
 for(Telefon telefon: telefonRepo.findAll()){
 str+=telefon.getSzám()+" "; "+telefon.getSzemély().getNév() + ";
 "+telefon.getSzemély().getJarmű().getSzín();
 str+="
";
 }
 return str;
}

String D(){
 Személy személy = személyRepo.findByNév("Kiss József");
 String str = személy.getNév() + " "; " + személy.getMagasság() + " "; " +
 személy.getJarmű().getMárka();
 return str;
}
}

```

## 9-10. gyakorlat - Spring-Boot – Security – Roles, JPA MySQL - ZH-BAN NEM LESZ, CSAK A BEADANDÓHOZ KELL

<https://spring.io/guides/gs/securing-web/>

<https://spring.io/projects/spring-security>

[https://www.tutorialspoint.com/spring\\_boot/spring\\_boot\\_securing\\_web\\_applications.htm](https://www.tutorialspoint.com/spring_boot/spring_boot_securing_web_applications.htm)

<https://www.marcobehler.com/guides/spring-security>

**Spring Security** is a powerful and highly customizable authentication and access-control framework. It is the de-facto standard for securing Spring-based applications.

**Spring Security** is a framework that focuses on providing both authentication and authorization to Java applications. Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements

### Features

- Comprehensive and extensible support for both Authentication and Authorization
- Protection against attacks like session fixation, clickjacking, cross site request forgery, etc
- Servlet API integration
- Optional integration with Spring Web MVC
- Much more...

<https://www.javaguides.net/2018/09/spring-boot-spring-mvc-role-based-spring-security-jpa-thymeleaf-mysql-tutorial.html>

alapján egyszerűsítve.

### Feladat

#### Felhasználói szerepek kezelése

##### **Egyszerűbb eset lenne - Nem ezt használjuk**

Minden felhasználóhoz csak egy szerep tartozhatna

pl. Admin, User, Látogató

Ilyenkor elég lenne csak a user tábla, amibe felvennénk egy mezőt pl. szerep és ebben tárolnánk:

pl. 1: regisztrált felhasználó (User), 2: admin (Admin)

##### **Bonyolultabb eseteket is támogató megoldás - Ezt használjuk**

##### **A Java Spring támogatja, hogy egy user-hez több szerep is tartozhat.**

Majd találkozunk a **UserDetails** interfésszel:

<https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetails.html>

<https://docs.spring.io/spring-security/site/docs/current/api/org/springframework/security/core/userdetails/UserDetails.html>

Ennek a **getAuthorities()** metódusa mutatja, hogy egy userhez több szerep is tartozhat:

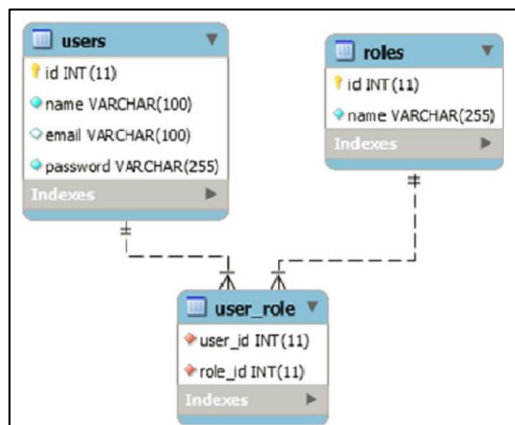
java.util.**Collection**<? extends GrantedAuthority> **getAuthorities()**

Returns the authorities granted to the user.

Mivel a Spring ezt a szerkezetet támogatja, ezért mi is ezt használjuk:

##### **Egy user-hez több szerep is tartozhat, egy szerephez több user is tartozhat:**

Ez egy M:N (több a többhöz) kapcsolatot jelent a userok és a szerepek között, ezért kell egy kapcsolótábla is:



A **user** és a **roles** táblák között **M:N (Many-to-Many)** kapcsolat van, amit a **user\_role** táblával oldunk meg.

Van még egy **messages** nevű táblánk is üzenetek tárolásához.

### XAMPP, MySQL, phpMyAdmin indítása

Importáljuk be a Forrásoknál lévő **feladatsec.sql** fájlt alapján az adatbázist

Email címmel lehet bejelentkezni.

#### Főoldal:

<http://localhost:8080/>

#### Látogató csak egy menüpontot lát:

Login

#### User 2 menüpontot lát:

User

Logout

#### Admin 3 menüpontot lát:

User

Logout

Admin

<http://localhost:8080/login>

Please sign in

Username

Password

Sign in

#### Bejelentkezések:

[admin@gmail.com](mailto:admin@gmail.com) password: jelszo1

[user@gmail.com](mailto:user@gmail.com) password: jelszo2

A **Kiegészítő feladat**-nál leírtam egy módszert, amivel lehet jelszóhoz BCrypt kódot generálni.

Csak bejelentkezett felhasználó láthatja a következő oldalt:

<http://localhost:8080/home>

Az oldalakon alkalmazzunk egy egyszerű vízszintes menüt.

Bejelentkezett felhasználónál kiírjuk a felhasználó email címét is.

[Home](#) [Logout](#)

Welcome user@gmail.com

**Csak bejelentkezett felhasználó láthatja ezt az oldalt.**

Ha adminként jelentkezünk be, akkor látható az Admin menü és elérhető az Admin oldal:

<http://localhost:8080/admin/home>

[Home](#) [Logout](#) [Admin](#)

Welcome admin@gmail.com

**Csak az Admin láthatja ezt az oldalt**

### Megoldás

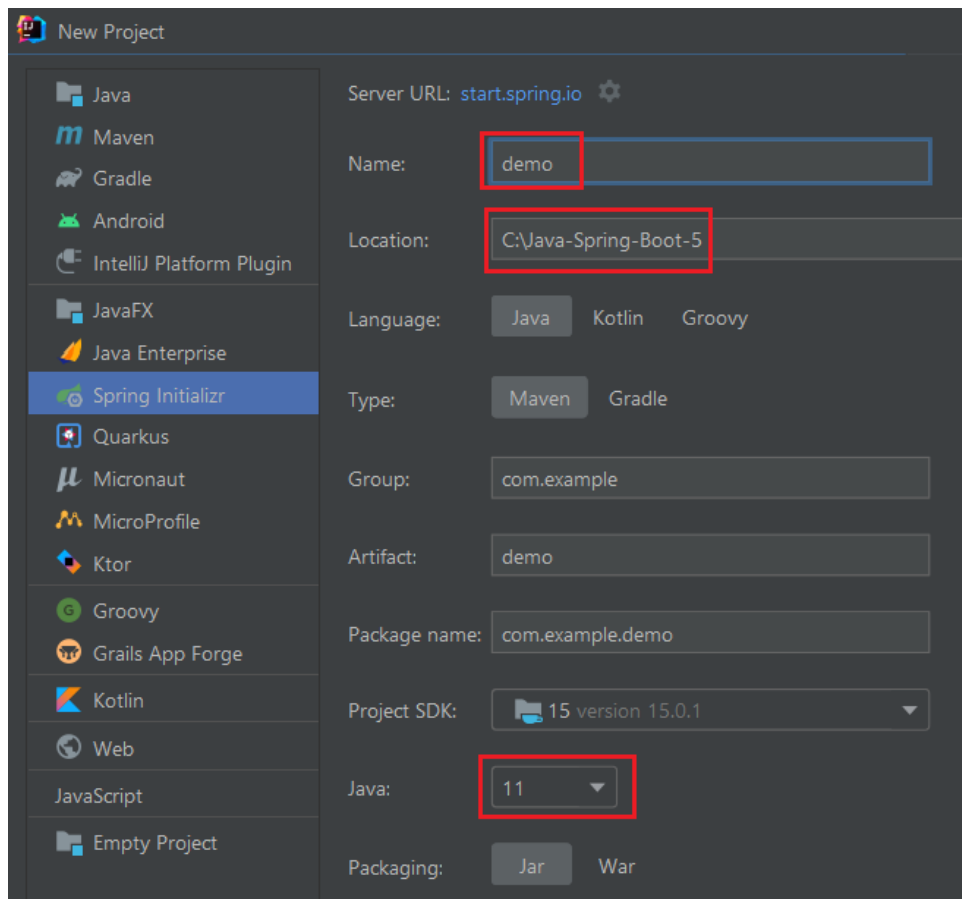
Készítsünk egy mappát a projektnek pl. c:\Java-Spring-Boot-5

### Előkészítés

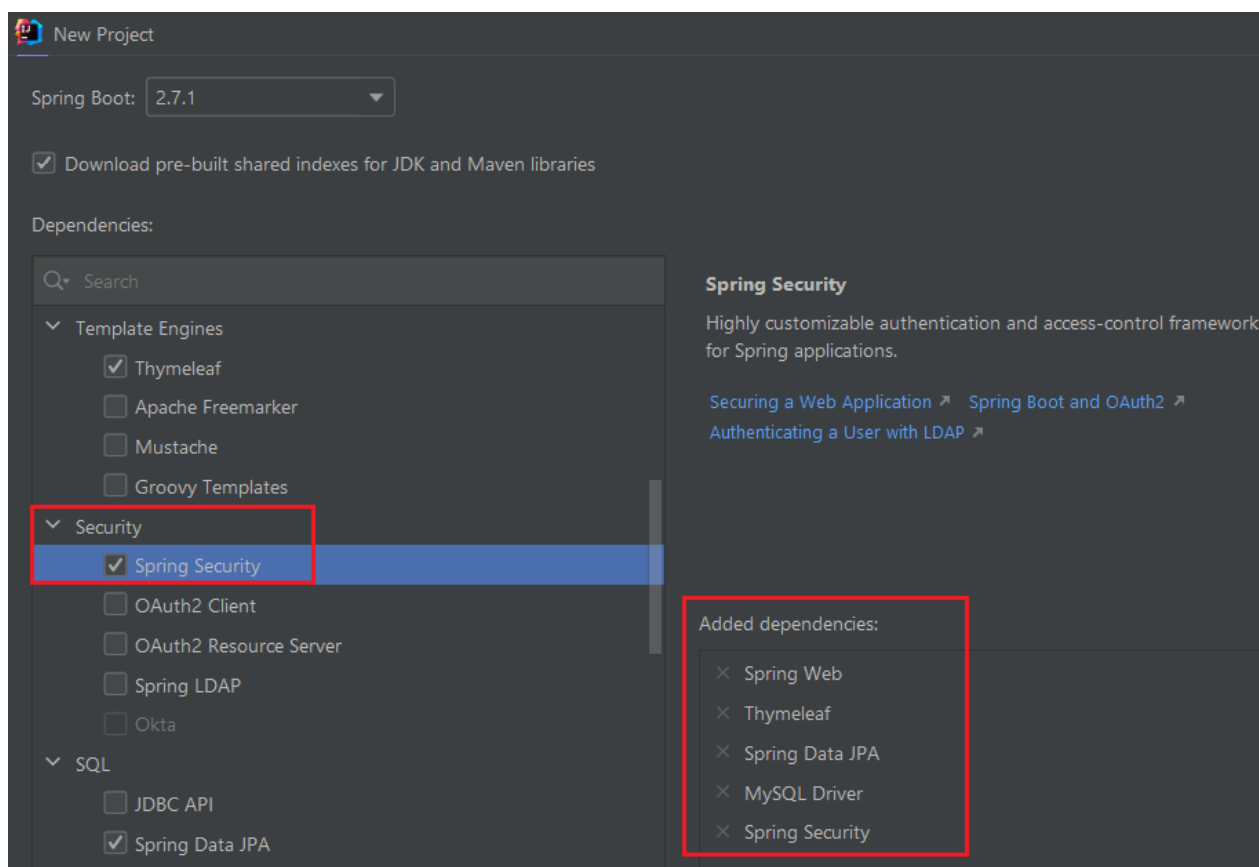
Töltsük le az alaprendszert: **Források.zip**-be is bemásolva

**A, módszer**

**IntelliJ / New project / Spring Initializr /**



Next



vagy

**B, módszer: volt amikor hibát adott!**

<https://start.spring.io/>

**Project**  
☒ Maven Project ☐ Gradle Project  
**Language**  
☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**  
☐ 3.0.0 (SNAPSHOT) ☐ 3.0.0 (M3) ☐ 2.7.2 (SNAPSHOT) ☒ 2.7.1  
☐ 2.6.10 (SNAPSHOT) ☐ 2.6.9

**Project Metadata**  
Group: com.example  
Artifact: demo  
Name: demo  
Description: Demo project for Spring Boot  
Package name: com.example.demo  
Packaging: ☒ Jar ☐ War  
Java: ☐ 18 ☐ 17 ☒ 11 ☐ 8

**Dependencies** ADD DEPENDENCIES... CTRL + B  
**Spring Web** **WEB**  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.  
**Spring Data JPA** **SQL**  
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.  
**MySQL Driver** **SQL**  
MySQL JDBC and R2DBC driver.  
**Thymeleaf** **TEMPLATE ENGINES**  
A modern server-side Java template engine for both web and standalone environments. Allows HTML to be correctly displayed in browsers and as static prototypes.  
**Spring Security** **SECURITY**  
Highly customizable authentication and access-control framework for Spring applications.

=> Generate

Nézzük meg a **pom.xml** fájlt.

**Ha hibákat jelez: File menü / Invalidate Caches...**

Töltsük ki az üres `src/main/resources/application.properties` fájlt:

`spring.datasource.url=jdbc:mysql://localhost:3306/feladatsec`

`spring.datasource.username=root`

`spring.datasource.password=`

`spring.jpa.properties.hibernate.format_sql=true`

**Ha nem aktív a Run gomb várakozási idő után sem:**

File menu / Invalidate Chaches ...

**Futtassuk az alkalmazást => Jól működik.**

Alap package: `com.example.demo`

**Nevezzük át (Refactor) `com.example.securityrole` -ra.**

**DemoApplication** osztályt nevezzük át (Refctor) **SecurityRoleApplication**-ra.

## 1. feladat

User, Role, Message JPA Entity-k

**Készítsük el a User JPA Entity-t:**

`package com.example.securityrole;`

`import javax.persistence.*;`

`import java.util.List;`

**@Entity**

**@Table(name="users")**

**public class User {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

```

private Integer id;
private String name;
private String email;
private String password;
@ManyToMany(cascade=CascadeType.MERGE)
@JoinTable(
 name="user_role",
 joinColumns={ @JoinColumn(name="USER_ID", referencedColumnName="ID")},
 inverseJoinColumns={ @JoinColumn(name="ROLE_ID", referencedColumnName="ID")})
private List<Role> roles;
}

```

**Generáltassuk a getter és setter metódusokat a végéhez: Code menü / Generate / Getter and Setter**

**Készítsük el a Role JPA Entity-t:**

```

package com.example.securityrole;
import javax.persistence.*;
import java.util.List;

```

```

@Entity
@Table(name = "roles")
public class Role {
 @Id
 @GeneratedValue(strategy = GenerationType.IDENTITY)
 private Integer id;
 @Column(nullable = false, unique = true)
 private String name;
 @ManyToMany(mappedBy = "roles")
 private List< User > users;
}

```

**Generáltassuk a getter és setter metódusokat a végéhez: Code menü / Generate / Getter and Setter**

## Repository-k

**Készítsük el: Spring Data JPA Repository Interface - UserRepository.java**

Ezzel könnyen elérjük a user-eket az adatbázisban.

A felhasználókat az email cím alapján keressük majd, pl. bejelentkezéskor.

```

package com.example.securityrole;
import java.util.Optional;
import org.springframework.data.repository.CrudRepository;
public interface UserRepository extends CrudRepository<User, Integer>
{
 Optional<User> findByEmail(String email); // email alapján lesz a bejelentkezés
}

```

Nem kell a roles táblához Repository: a User és Role osztályokban megadtuk a kapcsolatokat, ami alapján megtalálja az adott user-hez a szerepeket.

<https://docs.spring.io/spring-security/site/docs/3.0.x/apidocs/org/springframework/security/core/userdetails/UserDetailsService.html>

A Spring Security a **UserDetailsService** interfészt használja, ami tartalmazza a `loadUserByUsername(String username)` metódust. Ez megkeresi a `UserDetails`-t egy adott username-hez. A `UserDetails` interfész egy bejelentkezett user objektumot reprezentál. Implementáljuk a `UserDetailsService`-t, hogy megkapjuk a `UserDetails`-t az adatbázisból:

Készítsük el a **CustomUserDetailsService** osztályt:

```
package com.example.securityrole;
import java.util.Collection;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
public class CustomUserDetailsService implements UserDetailsService {
 @Autowired
 private UserRepository userRepo;
 @Override
 public UserDetails loadUserByUsername(String userName) throws UsernameNotFoundException {
 User user = userRepo.findByEmail(userName)
 .orElseThrow(() -> new UsernameNotFoundException("Email " + userName + " not found"));
// Egy új User-t (Security) hoz létre.
 return new org.springframework.security.core.userdetails.User(user.getEmail(), user.getPassword(),
 getAuthorities(user));
 }
// A kiválasztott felhasználó szerepeinek lekérdezése:
 private static Collection<? extends GrantedAuthority> getAuthorities(User user) {
 String[] userRoles = user.getRoles().stream().map((role) -> role.getName()).toArray(String[]::new);
 Collection<GrantedAuthority> authorities = AuthorityUtils.createAuthorityList(userRoles);
 return authorities;
 }
}
```

A security konfiguráció beállításához készítsük el a **WebSecurityConfig** osztályt:

```
package com.example.securityrole;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import
org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
```



```
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.web.util.matcher.AntPathRequestMatcher;
```

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(securedEnabled = true, proxyTargetClass = true)
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
 @Autowired
 private UserDetailsService customUserDetailsService;
 @Autowired
 public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception {
 auth.userDetailsService(customUserDetailsService).passwordEncoder(new
BCryptPasswordEncoder());
 }
 @Override
 protected void configure(HttpSecurity http) throws Exception {
 http.authorizeRequests()
 .antMatchers("/resources/**", "/").permitAll()
 .antMatchers("/admin/**").hasRole("ADMIN")
 .anyRequest().authenticated()
 .and()
 .formLogin().defaultSuccessUrl("/home").permitAll()
 .and()
 .logout().logoutRequestMatcher(new AntPathRequestMatcher("/logout"))
 .logoutSuccessUrl("/").permitAll()
 .and()
 .exceptionHandling();
 }
}
```

Konfiguráltuk a CustomUserDetailsService –t és a BCryptPasswordEncoder –t: az AuthenticationManager –t használja az alap in-memory adatbázis helyett.

#### A configure(HttpSecurity http) metódussal beállítottuk:

Engedélyezze mindenkinek a következő útvonalakat: "/resources/", "/"

Az /admin/ -al kezdődő útvonalakat csak az ADMIN szerepűek érhetik el.

Minden más URL-t csak bejelentkezett felhasználók érhetnek el.

Sikeres belépéskor a **"/home"** útvonalra irányítjuk.

Kijelentkezéskor a **"/"** útvonalra irányítjuk.

## Controller

#### Készítsük el a HomeController –t:

```
package com.example.securityrole;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
```

```
@Controller
```

```

public class HomeController {
 @GetMapping("/")
 public String home() {
 return "index";
 }
 @GetMapping("/home")
 public String user(Model model) {
 return "user";
 }
 @GetMapping("/admin/home")
 public String admin() {
 return "admin";
 }
}

```

## Nézetek Thymeleaf -el

src/main/resources/templates/**menu.html**

```

<div xmlns:th="http://www.thymeleaf.org"
 xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
 <div>

 <a th:href="@{/login}">Login

 <a th:href="@{/home}">Home
 <a th:href="@{/logout}">Logout

 <a th:href="@{/admin/home}">Admin

 </div>
 <div sec:authorize="isAuthenticated()">
 <h3>Welcome User</h3>
 </div>
</div>

```

src/main/resources/templates/**index.html**

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:th="http://www.thymeleaf.org">
 <head>
 <title>Home</title>
 </head>
 <body>
 <div th:insert="menu"></div>
 </body>
</html>

```

src/main/resources/templates/**user.html**

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"

```

```

 xmlns:th="http://www.thymeleaf.org">
<head>
 <title>User Home</title>
</head>
<body>
 <div th:insert="menu"></div>
 <h1>Csak bejelentkezett felhasználó láthatja ezt az oldalt.</h1>
</body>
</html>

```

```

src/main/resources/templates/admin.html
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:th="http://www.thymeleaf.org">
<head>
 <title>Admin Home</title>
</head>
<body>
 <div th:insert="menu"></div>
 <h1>Csak az Admin láthatja ezt az oldalt</h1>
</body>
</html>

```

## Próbáljuk ki a kész alkalmazást!

### Kiegészítő feladat - BCrypt kód generálás (nincs a megoldásban)

A következő módszerrel generáltam 2 jelszót, hogy ki tudjuk próbálni a bejelentkezést, és ezeket a jelszókat írtam az adatbázistáblákba:

jelszo1: \$2a\$10\$QEaf3I.eLiZC4F4pDnqmC.sTysFIJ59wgROmw3ATxceFs/wgg0LvK  
jelszo2: \$2a\$10\$exVjZOnYQ3oFdNTFP7qVHOoL8K2XhKpWXY3r8duw8v9pTNxmC0qbm  
Tehát az [admin@gmail.com](mailto:admin@gmail.com) jelszava **jelszo1**, a [user@gmail.com](mailto:user@gmail.com) jelszava: **jelszo2**

Készítsünk egy oldalt, ami egy adott jelszóhoz kiírja a BCrypt kódot

```

@GetMapping("/jelszoteszt")
@ResponseBody
public String jelszoTeszt() {
 BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();
 return bCryptPasswordEncoder.encode("jelszo1");
}

```

A **WebSecurityConfig** osztályban engedélyezni kell, hogy bárki elérhesse a **/jelszoteszt** útvonalat.

```

protected void configure(HttpSecurity http) throws Exception {
 http.authorizeRequests()
 .antMatchers("/resources/**", "/", "/jelszoteszt").permitAll()

}

```

## 2. feladat: Kiegészítés regisztrációs oldallal

**HomeController.java** folytatása - új részek pirossal jelölve

```
package com.example.securityrole;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

import java.util.ArrayList;
import java.util.List;

@Controller
public class HomeController {
 @GetMapping("/")
 public String home() {
 return "index";
 }
 @GetMapping("/home")
 public String user(Model model) {
 return "user";
 }
 @GetMapping("/admin/home")
 public String admin() {
 return "admin";
 }

 @GetMapping("/regisztral")
 public String greetingForm(Model model) {
 model.addAttribute("reg", new User());
 return "regisztral";
 }

 @Autowired
 private UserRepository userRepo;
 @PostMapping("/regisztral_feldolgoz")
 public String Regisztráció(@ModelAttribute User user, Model model) {
 for(User felhasználó2: userRepo.findAll())
 if(felhasználó2.getEmail().equals(user.getEmail())){
 model.addAttribute("uzenet", "A regisztrációs email már foglalt!");
 return "reghiba";
 }
 BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
 user.setPassword(passwordEncoder.encode(user.getPassword()));
 Role role = new Role();
 // Minden regisztrációkor USER szerepet adunk a felhasználónak:
 role.setId(3); role.setName("ROLE_USER");
 List<Role> rolist = new ArrayList<Role>();
 rolist.add(role);
 user.setRoles(rolist);
 }
}
```

```

 userRepo.save(user);
 model.addAttribute("id", user.getId());
 return "regjo";
 }
}

```

src/main/resources/templates/**regisztral.html**

```

<!DOCTYPE HTML>
<html xmlns:th="https://www.thymeleaf.org">
<head>
 <title>Bejelentkezés, regisztráció</title>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
 <div th:insert="menu"></div>
 <h3>Regisztrálja magát, ha még nem felhasználó!</h3>
 <form action="#" th:action="@{/regisztral_feldolgoz}" th:object="${reg}" method="post">
 <fieldset>
 <legend>Regisztráció</legend>
 <p><input type="text" th:field="*{name}" placeholder="Név" required/></p>
 <p><input type="text" th:field="*{email}" placeholder="Email" required/></p>
 <p><input type="password" th:field="*{password}" placeholder="Jelszó" required/></p>
 <p><input type="submit" value="Regisztráció" /></p>
 </fieldset>
 </form>
</body>
</html>

```

src/main/resources/templates/**reghiba.html**

```

<!DOCTYPE html>
<html xmlns:th="https://www.thymeleaf.org">
<html lang="en">
<head>
 <meta charset="UTF-8">
 <title>Regisztrációs hiba</title>
</head>
<body>
 <div th:insert="menu"></div>
 <h1 th:text="${uzenet}" />
 Próbálja újra.
</body>
</html>

```

src/main/resources/templates/**regjo.html**

```

<!DOCTYPE HTML>
<html xmlns:th="https://www.thymeleaf.org">
<head>
 <title>Sikeres bejelentkezés</title>
 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
</head>
<body>
 <div th:insert="menu"></div>
 <h1 th:utext="A regisztrációja sikeres.
 Azonosítója: ' + ${id} " />

```

```
</body>
</html>
```

Egészítsük ki a src\main\resources\templates\menu.html oldalt:

```
.....

 <a th:href="@{/regisztral}">Regisztrál

.....
```

Egészítsük ki a WebSecurityConfig.java fájlt:

Adunk mindenkinek jogot a /regisztral és /regisztral\_feldolgoz útvonalakhoz is:

```
.....
.antMatchers("/resources/**", "/", "/regisztral", "/regisztral_feldolgoz").permitAll()
.....
```

**Próbáljuk ki:**

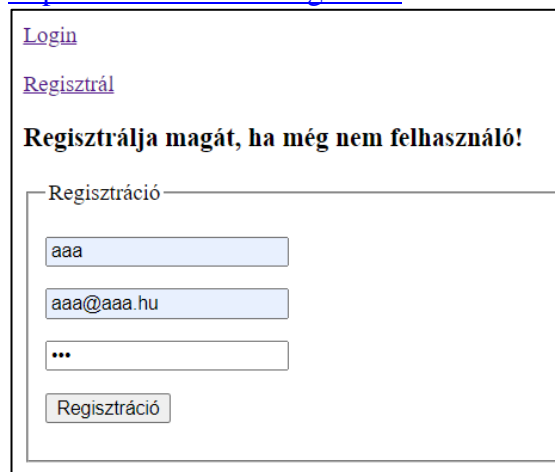
<http://localhost:8080/>

Login

Regisztrál

**Regisztráció:**

<http://localhost:8080/regisztral>



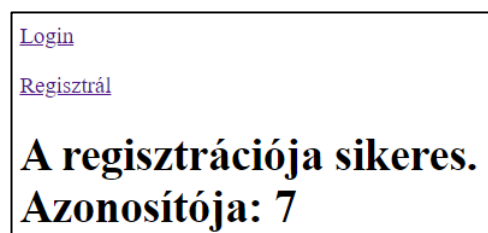
Jelszó pl. aaa

**Kiegészítő gyakorló feladat:**

A jelszót kétszer kelljen beírni. Csak akkor engedje tovább, ha a két jelszó megegyezik.

**Sikeres regisztráció után:**

[http://localhost:8080/regisztral\\_feldolgoz](http://localhost:8080/regisztral_feldolgoz)



Regisztráció után nézzük meg a **users** és a **user\_role** táblákat.

**Regisztráció, ha a regisztrációs email már foglalt:**

Elég az email egyezőségét vizsgálni!

[http://localhost:8080/regisztral\\_feldolgoz](http://localhost:8080/regisztral_feldolgoz)

[Login](#)

[Regisztrál](#)

**A regisztrációs email már foglalt!**

[Próbálja újra.](#)

**Jelentkezzünk be a most regisztrált felhasználóval:**

<http://localhost:8080/home>

[Home](#) [Logout](#)

Welcome aaa@aaa.hu

**Csak bejelentkezett felhasználó láthatja ezt az oldalt.**

# 11. gyakorlat - Spring-Boot – RESTful API - JPA MySQL

## Bevezetés - Csak olvasmány - Kihagyható

REST, RESTful

<https://hu.wikipedia.org/wiki/REST>

A **REST (Representational State Transfer)** egy szoftverarchitektúra típus, elosztott kapcsolat (loose coupling), nagy, internet alapú rendszerek számára.

Azokat a rendszereket, amelyek eleget tesznek a REST megszorításainak, "**RESTful**"-nak nevezik.

A REST architektúra típust párhuzamosan fejlesztették a HTTP specifikáció 1.1-es változatával.

A legnagyobb olyan rendszer, amely eleget tesz a REST szoftverarchitektúra típus követelményeinek a világháló. A REST szemlélteti a világháló architektúráját azzal, hogy leírja és megköti a világháló négy komponensének (kiszolgálók, átjárók, proxyk és kliensek) magas szintű kölcsönhatásait.

Egy **REST** típusú architektúra kliensekből és szerverekből áll. A kliensek kéréseket indítanak a szerverek felé; a szerverek kéréseket dolgoznak fel és a megfelelő választ küldik vissza. A kérések és a válaszok erőforrás-reprezentációk szállítása köré épülnek. Egy erőforrás-reprezentáció általában egy dokumentum, mely rögzíti az erőforrás jelenlegi vagy kívánt állapotát.

A **REST eredetileg a HTTP keretein belül lett leírva**, de nem korlátozódik erre a protokollra. Egy "**RESTful**" architektúra más alkalmazási rétegbeli protokollra is épülhet, amennyiben az már rendelkezik értelmes erőforrás-reprezentáció átvitelhez szükséges gazdag és egységes szókincssel.

A HTTP nagyon gazdag szókincssel rendelkezik igék (vagy "metódusok"), URI-k, média típusok, kérés- és feleletkódok stb. szempontjából. **Egy REST alkalmazás a HTTP protokoll meglévő tulajdonságait használja** és így lehetővé teszi a proxyknak és az átjáróknak, hogy együttműködjenek az alkalmazással (például gyorsítótárazás vagy biztonsági funkciók formájában).

[https://vik.wiki/8.\\_REST\\_\(2012\)](https://vik.wiki/8._REST_(2012))

### RESTful HTTP

- HTTP protokoll kibővítése: GET, POST, PUT, DELETE
- Bemenő paraméterek: URL része, URL query string, POST paraméter, HTTP body
- Visszatérési érték: HTTP body
- nagyon egyszerű: böngészőből is tesztelhető

### REST alapelvei

- Minden erőforráshoz azonosító rendelése (URI, URN, URL a jó, mert egyértelmű, könnyű feloldani, független a mögöttes technológiától)  
Az **URI** (Uniform Resource Identifier, egységes erőforrás-azonosító) egy rövid karaktersorozat, amelyet egy webes erőforrás azonosítására használunk. Közismert példái a webcímek, más néven **URL**-ek. Az **URI** az erőforrást kétféleképp azonosíthatja: hely szerint (**URL**) vagy név szerint (**URN**). Az URL olyan URI, amely azzal határozza meg az erőforrást, miképp lehet azt elérni. Például <https://hu.wikipedia.org/>. Az **URN**-re példa lehet a következő: urn:isbn:0-395-36341-1. Ez egy olyan URI, amely egy könyvet azonosít az ISBN-adata alapján. Ezzel az URN-nel azonosítottuk a könyvet anélkül, hogy bármit mondtunk volna a helyéről vagy az elérhetőségéről.
- Erőforrások lehetnek: doksik, adatok (számítás eredménye), szolgáltatások (SOAP, metaadatok, stb), fogalmak
- Dolgok összekapcsolása (jó URL címet kell választani)



- CRUD műveletek használata (Create, Read, Update, Delete)  
pl. egy adatbázis rekordhoz: Create, Read, Update, Delete  
pl. fájlfeltöltéshez: Create, Read, Update, Delete
- Állapotmentes kommunikáció

### Többféle adatrepresentáció

- HTML (emberek számára) vagy XML, JSON, stb. gépek számára
- változhat a struktúra

### Állapotmentes kommunikáció

- A REST önmagában állapotmentes
- De az alkalmazásnak lehet állapota (kliens oldalon, erőforrásban tárolva)
- Skálázhatósági előnyök emiatt (nincs session, felcserélhető szerverek)

<https://docs.microsoft.com/hu-hu/azure/architecture/best-practices/api-design>

A legtöbb modern webalkalmazás API (Application Programming Interface)-kat tesz elérhetővé, amelyek segítségével az ügyfelek interakcióba léphetnek az alkalmazással. Egy jól megtervezett webes API-nak a következők támogatására kell törekednie:

- **Platformfüggetlenség.** Bármelyik ügyfélnek meg kell tudnia hívni az API-t – függetlenül az API belső implementálásától. Ehhez szabványos protokollokra van szükség, valamint olyan mechanizmusra, amely által az ügyfél és a webszolgáltatás meg tud egyezni a kicserélendő adatok formátumában.
- **Szolgáltatásfejlődés.** A webes API-nak képesnek kell lennie a fejlődésre és új funkciók hozzáadására – az ügyfélalkalmazásoktól függetlenül. Az API fejlődése mellett biztosítani kell a meglévő ügyfélalkalmazások módosítás nélküli működését. Minden funkciónak felderíthetőnek kell lennie, hogy az ügyfélalkalmazások teljes mértékben használhassák azt.

A **REST** a webes szolgáltatások tervezésére szolgáló architekturális módszer. A REST egy architekturális stílus a hipermédián alapuló elosztott rendszerek készítéséhez. A REST mindennemű mögöttes protokolltól független, és nem feltétlenül kötődik a HTTP-hez. A leggyakoribb REST-alkalmazások azonban a HTTP-protokollt használják.

A REST elsődleges előnye, hogy nyílt szabványokat használ, és nem köti az API vagy az ügyfélalkalmazások megvalósítását semmilyen konkrét megvalósításhoz. Egy REST-alapú webszolgáltatás például megírható ASP.NET-ben, az ügyfélalkalmazások pedig bármilyen nyelvet vagy eszközkészletet használhatnak, amelyekkel HTTP-kérések hozhatók létre és HTTP-válaszok elemezhetők.

### RESTful API-k fő tervezési alapelvei közül néhány:

- A REST API-k erőforrások köré vannak szervezve. Az erőforrások olyan objektumok, adatok vagy szolgáltatások, amelyek az ügyfél által elérhetők.
- Minden erőforrás rendelkezik egy azonosítóval. Ez az URI, amely egyedileg azonosítja az adott erőforrást. Például egy adott ügyfélrendelés URI-ja a következő lehet:  
<https://adventure-works.com/orders/1>
- Az ügyfelek az erőforrások reprezentációinak cseréje révén lépnek interakcióba a szolgáltatásokkal. Számos webes API a JSON-t használja csereformátumként. Például, ha a fent említett URI-ra egy GET-kérés érkezik, akkor a rendszer a következő válaszüzenetet adhatja vissza: JSON:  
{ "orderId":1,"orderValue":99.90,"productId":1,"quantity":1 }
- A REST API-k egységes felületet használnak, amely segít az ügyfél és a szolgáltatás implementálásának különválasztásában. A HTTP-re épülő REST API-k esetében az egységes felület szabványos HTTP-műveleteket is tartalmaz az erőforrásokon végzett műveletek végrehajtásához. A leggyakoribb műveletek a következők: GET, POST, PUT, PATCH és DELETE.
- A REST API-k állapot nélküli kérésmodellt használnak. A HTTP-kéréseknek függetlennek kell lenniük, és bármilyen sorrendben előfordulhatnak, ezért nem valósítható meg az átmeneti állapotadatok kérések közötti megőrzése. Az információt egyedül maguk az erőforrások tárolják,

és minden kérésnek atomi műveletnek kell lennie. Ez a megkötés teszi lehetővé a webes szolgáltatások kiváló méretezhetőségét, mert nincs szükség az ügyfelek és kiszolgálók közötti affinitás megőrzésére. Bármely kiszolgáló képes kezelni bármilyen ügyféltől beérkező kérést.

- A REST API-kat a reprezentációban szereplő hipermédia-hivatkozások vezérlik. A következő példában egy rendelés JSON-reprezentációja látható. Hivatkozásokat tartalmaz, amelyek lekérdezik vagy frissítik a rendeléshez társított ügyfelet. JSON:

```
{
 "orderId":3,
 "productId":2,
 "quantity":4,
 "orderValue":16.60,
 "links": [
 { "rel":"product","href":"https://adventure-works.com/customers/3", "action":"GET" },
 { "rel":"product","href":"https://adventure-works.com/customers/3", "action":"PUT" }
]
}
```

### Műveletek meghatározása HTTP-metódusok keretében

A RESTful webes API-k által használt gyakoribb HTTP-metódusok a következők:

- GET: lekéri az erőforrás reprezentációját a megadott URI-n keresztül. A válaszüzenet törzse tartalmazza a kért erőforrás részleteit.
- POST: egy új erőforrást hoz létre a megadott URI-n. A kérésüzenet törzse tartalmazza az új erőforrás részleteit. Vegye figyelembe, hogy a POST olyan műveletek aktiválására is használható, amelyek nem hoznak létre erőforrásokat.
- PUT: A megadott URI-n létrehoz egy új erőforrást, vagy cseréli a meglévőt. A kérésüzenet törzse meghatározza a létrehozni vagy frissíteni kívánt erőforrást.
- PATCH: egy erőforrás részleges frissítését hajtja végre. A kérés törzse megadja az erőforrásra alkalmazni kívánt módosításokat.
- DELETE: eltávolítja az erőforrást a megadott URI-n.

### Rövidebben:

- a GET metódust erőforrás lekérésére használjuk.
- a POST metódust erőforrás létrehozására használjuk.
- a PUT metódust az erőforrás vagy az állapotának a módosítására használjuk.
- a DELETE metódust egy erőforrás törlésére, megszüntetésére használjuk.

### REST, RESTful, cURL, fake REST API, Tesztelés cURL-el, Postman-el

A cURL sok fajta protokollt tud kezelni:

<https://curl.haxx.se/docs/manpage.html>

curl is a tool to transfer data from or to a server, using one of the supported protocols (DICT, FILE, FTP, FTPS, GOPHER, HTTP, HTTPS, IMAP, IMAPS, LDAP, LDAPS, MQTT, POP3, POP3S, RTMP, RTMPS, RTSP, SCP, SFTP, SMB, SMBS, SMTP, SMTPS, TELNET and TFTP). The command is designed to work without user interaction. curl offers a busload of useful tricks like proxy support, user authentication, FTP upload, HTTP post, SSL connections, cookies, file transfer resume, Metalink, and more.

<https://hu.wikipedia.org/wiki/CURL>

A cURL egy számítógépes szoftver projekt, amely több protokollon keresztüli fájllelérést biztosít egy parancssori eszköz és egy könyvtár segítségével. A **cURL projekt** két terméket hoz létre, a **libcurl**-t és a **cURL**-t. Először 1997-ben adták ki.

#### **libcurl**

Egy ingyenes kliensoldali URL-transzfer könyvtár, amely támogatja az FTP, FTPS, Gopher, HTTP, HTTPS, SCP, SFTP, TFTP, Telnet, DICT, fájl URI-séma, LDAP, LDAPS,

IMAP, POP3, SMTP és RTSP protokollokat. A könyvtár támogatja a HTTPS tanúsítványokat, a HTTP POST-ot, a HTTP PUT-ot, az FTP-feltöltést, Kerberost, a HTTP űrlap alapú feltöltést, a proxykat, a sütiket, a felhasználónév-jelszóval történő autentikációt, a fájltranszfer-folytatást, és a HTTP proxyt.

A libcurl könyvtár hordozható. Több platformon működik ugyanúgy, beleértve a Solaris, NetBSD, FreeBSD, OpenBSD, Darwin, HP-UX, IRIX, AIX, Tru64, Linux, UnixWare, HURD, Windows, Symbian, Amiga, OS/2, BeOS, Mac OS X, Ultrix, QNX, BlackBerry Tablet OS,[2] OpenVMS, RISC OS, Novell NetWare, DOS és egyéb operációs rendszereket is.

A libcurl könyvtár ingyenes, szálbiztos, IPv6 kompatibilis és gyors. A libcurl-höz több, mint 40 programnyelven érhető el csatolás.

## cURL

Parancssori eszköz a fájlok URL-szintaxissal való lekérésére vagy elküldésére.

Windows parancssorba is működik pl. curl --help

Mivel a cURL libcurl-t használ, így nagy mennyiségű protokollt támogat, így például a HTTP-t, a HTTPS-t, az FTP-t, az FTPS-t, az SCP-t, az SFTP-t, a TFTP-t, az LDAP-ot, az LDAPS-t, a DICT-et, a TELNET-et, a FILE-t, az IMAP-ot, a POP3-at, az SMTP-t és az RTSP-t (az utolsó négyet kizárólagosan a 7.20.0 vagy 2010. február 9. óta).

A projekt nevének eredte a "Client for URLs" (kliens az URL-eknek), eredetileg nagybetűsen írva az URL-t, annak egyértelműsítésére, hogy URL-eket kezel. A tény, hogy kiejthető, mint "see URL" (lásd az URL-t) szintén segített; a "Client URL Request Library" (Kliens az URL-lekérő könyvtárnak) és a rekurzív "Curl URL Request Library" (CURL URL-lekérő könyvtár) rövidítéseként is működik.

Daniel Stenberg a cURL írását 1997-ben kezdte, egy több-protokollon, például a HTTP-n, az FTP-n, a GOPHER-en, és egyéb protokollokon keresztüli fájlátviteli parancssoros programként. Több egyéb ember is fontos vagy életbevágó változtatást hajtott végre a projekten. A cURL szabadszoftver, az MIT License alatt lett kiadva.

<https://www.baeldung.com/curl-rest>

cURL is a command-line tool for transferring data and supports about 22 protocols including HTTP. This combination makes it a very good ad-hoc tool for **testing our REST services**.

A legtöbb operációs rendszer tartalmazza a cURL-t, így a Windows is és parancssorból is használható. Először ezt próbáljuk ki.

Nyissa meg a parancssort a Windows-ban.

A cURL-ben lehet használni a GET, POST, PUT és DELETE metódusokat is.

**A GET metódus használata a cURL-ben.** pl: a parancssorba beírni:

curl <https://index.hu> letölti a HTML oldalt.

curl -o proba.txt <https://index.hu> A HTML oldalt letölti a proba.txt fájlba.

## Tesztelés cURL-el

**Több olyan fake REST API van, ahol lehet tesztelni a metódusokat.**

<https://gorest.co.in>

<https://jsonplaceholder.typicode.com/>

<https://reqres.in/>

<https://dummyapi.io/>

<https://myfakeapi.com/>

<https://fakestoreapi.com/>

<https://dummy.restapiexample.com/>

Ezek közül az egyik: <https://gorest.co.in/>

Regisztrálni kell a használathoz, mert kell az Access Token (Authorization kód).

Regisztráljanak és olvassák ki az Authorization kódot (Access Token)

**Access Token:** \*\*\*\*\*

Ez minden felhasználónál egyedi. Ezt kell majd beírni a Authorization: rész után.

### GET módszer használata:

Ez még nem jó, hibát jelez: curl https://gorest.co.in/public-api/users

curl -H "Authorization: Bearer \*\*\*\*\*" <https://gorest.co.in/public-api/users>

### Válasz:

```
{ "code":200,"meta":{"pagination":{"total":3210,"pages":321,"page":1,"limit":10}}, "data":[{"id":3310,"name":"Herbert Steuber","email":"herbert_steuber@corkery.net","gender":"female","status":"inactive"}, {"id":3293,"name":"Bart Windler LLD","email":"bart_ild_windler@mcglynn.co","gender":"female","status":"inactive"},] }
```

A Válaszban a "data" résznél megnézni, hogy milyen mezőneveket használ. Pirossal bejelöltem felül. Ezeket kell használni a következő műveleteknél.

### POST módszer használata:

**Ez lenne az alaputasítás:**

```
curl -i -H "Accept:application/json" -H "Content-Type:application/json" -H "Authorization: Bearer *****" -XPOST "https://gorest.co.in/public-api/users" -d '{"name":"Nagy Tibor","gender":"male","email":"proba@data.hu","status":"active"}'
```

de a Windows-ban ez hibás, mert nem fogadja el az egyes aposztróf (') karaktert. Ezért kettőse kell átírni és a belső helyeken (") helyett (\"). Így már jó lesz:

```
curl -i -H "Accept:application/json" -H "Content-Type:application/json" -H "Authorization: Bearer *****" -XPOST "https://gorest.co.in/public-api/users" -d '{"name\\":\\"Nagy Tibor\\",\\"gender\\":\\"male\\",\\"email\\":\\"proba@data.hu\\",\\"status\\":\\"active\\"}'
```

### Sikeres! Ezt írja vissza:

```
.....
location: https://gorest.co.in/public-api/users/4903
```

```
.....
{ "code":201,"meta":null,"data":{"id":4903,"name":"Nagy Tibor","email":"proba@data.hu","gender":"male","status":"active"} }
```

### A **4903** az egyedi azonosító.

### GET módszerrel ellenőrizni a felvitt adatokat:

Így nagyon sok adatot ad:

```
curl -H "Authorization: Bearer *****" https://gorest.co.in/public-api/users
```

### A **4903-es ID** lekérdezése:

```
curl -H "Authorization: Bearer *****" https://gorest.co.in/public-api/users/4903
```

Kiírja az adatokat.

### PUT módszer (módosítás) használata:

Írjuk át a name adatot Nagy Tibor-ról Kiss Edit-re:

```
curl -i -H "Accept:application/json" -H "Content-Type:application/json" -H "Authorization: Bearer *****" -X PUT "https://gorest.co.in/public-api/users/4903" -d '{"name\\":\\"Kiss Edit\\",\\"gender\\":\\"male\\",\\"email\\":\\"proba@data.hu\\",\\"status\\":\\"active\\"}'
```

Sikeresen átírta!

Ellenőrzés:

```
curl -H "Authorization: Bearer *****" https://gorest.co.in/public-api/users/4903
```

**Elég csak az átírandó mezőt megadni:**

Írjuk át a name adatot Toth Laszlo-ra

```
curl -i -H "Accept:application/json" -H "Content-Type:application/json" -H "Authorization: Bearer *****" -X PUT "https://gorest.co.in/public-api/users/4903" -d {"name":"Toth Laszlo"}
```

Ellenőrzés:

```
curl -H "Authorization: Bearer *****" https://gorest.co.in/public-api/users/4903
```

**DELETE metódus használata:**

```
curl -i -H "Accept:application/json" -H "Content-Type:application/json" -H "Authorization: Bearer *****" -X DELETE https://gorest.co.in/public-api/users/4903
```

Sikeresen törölte!

**GET metódussal ellenőrizni a törlést:**

```
curl -H "Authorization: Bearer *****" https://gorest.co.in/public-api/users/4903
```

Már nem található!

*Tesztelés Postman-el*

<https://www.postman.com/>

<https://www.postman.com/downloads/>

Portable (nem kell telepíteni):

<https://portapps.io/app/postman-portable/>

Online:

<https://www.postman.com/downloads/>

**Töltsük le a Portable verziót.**

Indítsuk el az exe fájlt => Kicsomagolja a kért mappába.

Indítsuk el az alkalmazást.

Nem kell account-ot regisztrálni.

<https://learning.postman.com/docs/getting-started/introduction/>

<https://learning.postman.com/docs/getting-started/sending-the-first-request/>

<https://learning.postman.com/docs/sending-requests/requests/>

**Create a request**

Scratch Pad / New

=> HTTP Request

Scratch Pad

New

**GET**

Ide még nem kell Authorization.

<https://gorest.co.in/public-api/users>

GET	▼	<a href="https://gorest.co.in/public-api/users">https://gorest.co.in/public-api/users</a>	Send	▼
-----	---	-------------------------------------------------------------------------------------------	------	---

=> Send

megadja JSON formában az eredményt:

Pretty:

```
{
 "id": 2603,
 "name": "Hamsini Somayaji",
 "email": "somayaji_hamsini@carroll.net",
 "gender": "male",
 "status": "inactive"
},
{
 "id": 2602,
 "name": "Bhadrak Jha Jr.",
 "email": "jr_bhadrak_jha@gleason.info",
 "gender": "female",
 "status": "active"
},
.....
```

Raw:

```
{"code":200,"meta":{"pagination":{"total":2592,"pages":260,"page":1,"limit":10}},"data":[{"id":2603,"name":"Hamsini Somayaji","email":"somayaji_hamsini@carroll.net","gender":"male","status":"inactive"},{"id":2602,"name":"Bhadrak Jha Jr.","email":"jr_bhadrak_jha@gleason.info","gender":"female","status":"active"},
.....
```

**POST**

<https://www.toolsqa.com/postman/post-request-in-postman/>

<https://www.javatpoint.com/post-request-in-postman>

Innentől már kell Authorization.

```
{
 "name": "Nagy Tibor",
 "gender": "male",
 "email": "proba@data.hu",
 "status": "active"
}
```

POST ▼ https://gorest.co.in/public-api/users

Params **Authorization** ● Headers (9) Body ● Pre-request Script Tests Settings

Type Bearer Token ▼

! Heads up! These parameters hold sensitive data. To keep this data secure while working in a collabor recommend using variables. [Learn more about variables](#) ↗

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#) ↗

Token 526b08bfb1b072683fdf288a758e007e6c5'...

POST ▼ https://gorest.co.in/public-api/users

Params Authorization ● Headers (9) **Body** ● Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL JSON ▼

```

1 {
2 "name": "Nagy Tibor",
3 "gender": "male",
4 "email": "proba@data.hu",
5 "status": "active"
6 }
```

## Válasz:

Body Cookies Headers (26) Test Results

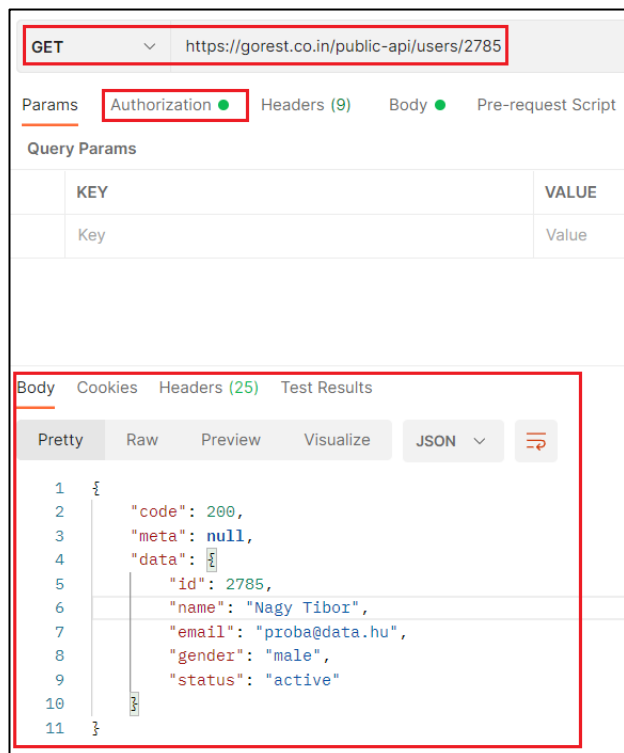
Pretty Raw Preview Visualize JSON ▼ ↺

```

1 {
2 "code": 201,
3 "meta": null,
4 "data": {
5 "id": 2785,
6 "name": "Nagy Tibor",
7 "email": "proba@data.hu",
8 "gender": "male",
9 "status": "active"
10 }
11 }
```

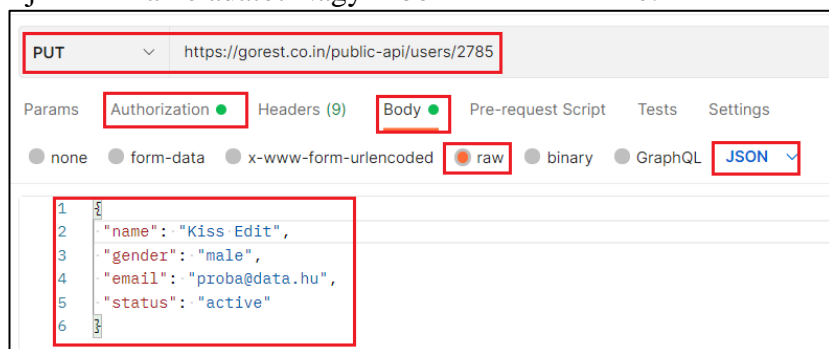
A **2785** az egyedi azonosító.

GET metódussal ellenőrizni a felvitt adatot:



### PUT metódus (módosítás) használata:

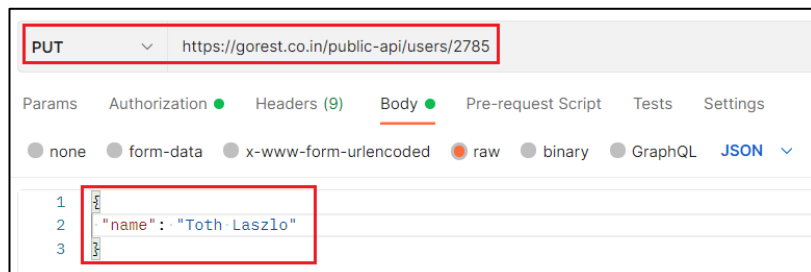
Írjuk át a name adatot Nagy Tibor-ról Kiss Edit-re:



GET metódussal ellenőrizni a módosítást: jó!

### Elég csak az átírandó mezőt megadni:

Írjuk át a name adatot Toth Laszlo-ra



GET metódussal ellenőrizni a módosítást: jó!

### DELETE metódus használata:



DELETE	https://gorest.co.in/public-api/users/2785
Params	Authorization ● Headers (9) Body ● Pre-request Script Tests Settings
Query Params	
KEY	VALUE
Key	Value

**GET metódussal ellenőrizni a törlést: már nincs meg a rekord!**

## A feladat

2. Készítsünk szerver oldali alkalmazást, amely a következőképpen szolgálja ki a kérélmeket:

- Paraméter nélküli **GET** kérelem: visszaadja az adatbázistábla adatait
- Paraméteres (ID) GET kérelem: visszaadja az adott ID-jű személy adatait
- POST** kérelem esetén létrehoz egy új sort az adatbázistáblában
- PUT** kérelem esetén módosítja a kapott azonosítóval (id paraméter) rendelkező személy adatait.
- DELETE** kérelem esetén törli a kapott azonosítóval (id paraméter) rendelkező személyt.

## Megoldás

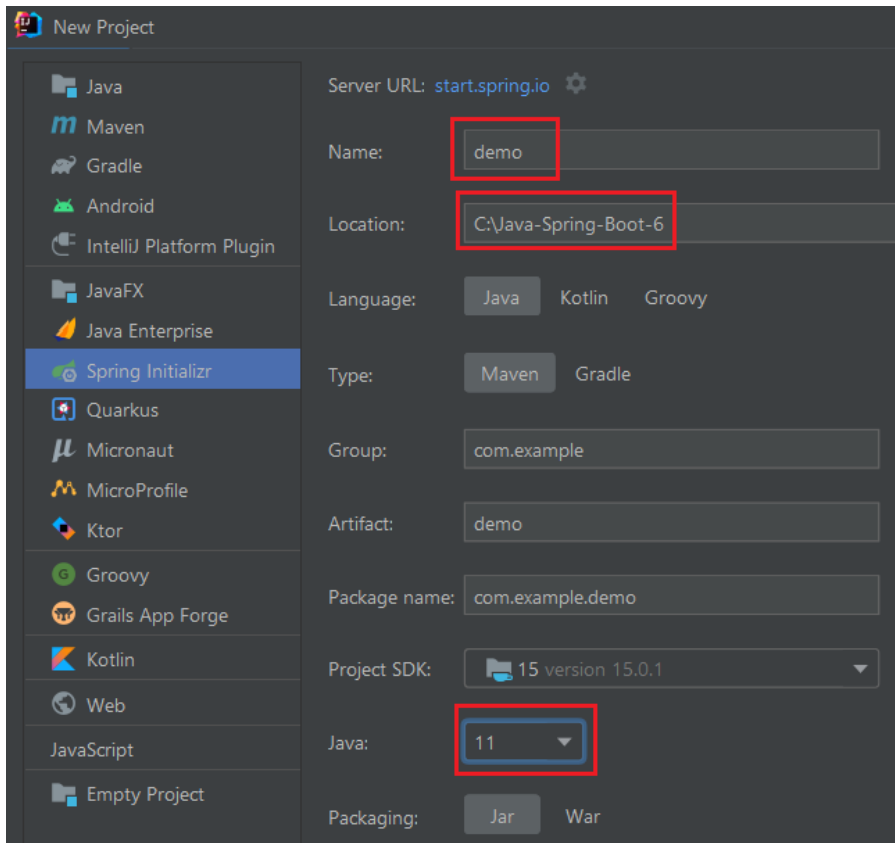
Készítsünk egy mappát a projektnek pl. **c:\Java-Spring-Boot-6**

## Előkészítés

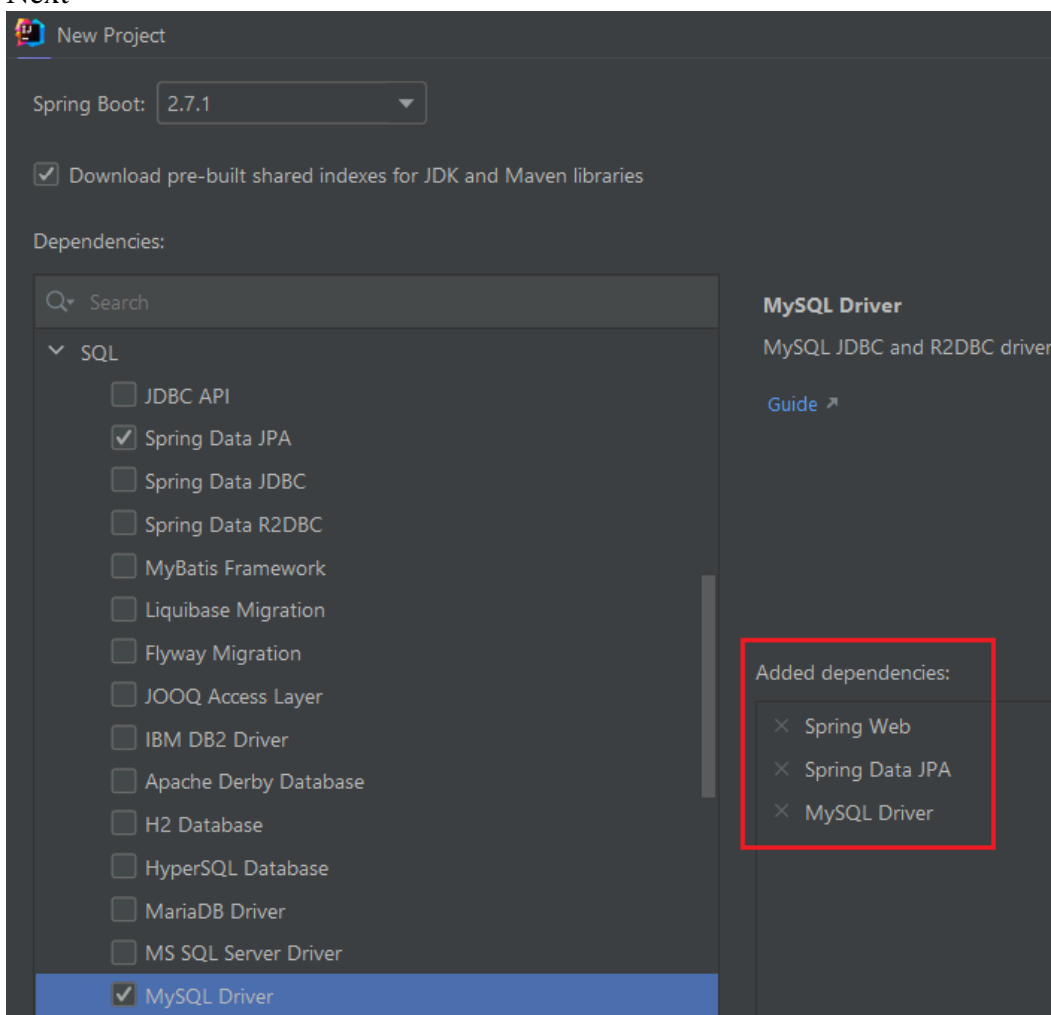
**Források.zip** –be is bemásoltam.

**A, módszer**

**IntelliJ / New project / Spring Initializr /**



Next



Finish

vagy

**B, módszer: volt amikor hibát adott!**

<https://start.spring.io>

Dependencies

ADD DEPENDENCIES... CTRL + B

Spring Web

WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA

SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

MySQL Driver

SQL

MySQL JDBC and R2DBC driver.

Nézzük meg a **pom.xml** fájlt.

**Ha hibákat jelez: File menü / Invalidate Caches...**

**Ha nem aktív a Run gomb várakozási idő után sem:**

File menü / Invalidate Caches...

Alapcsomag: package com.example.demo;

**Nevezzük át (Refactor):** com.example.gyakorlat;

Kiinduló osztály: DemoApplication

**Nevezzük át (Refactor):** RESTfulFeladat

Indítsuk a **XAMPP**-ot. Importáljuk be a **restgyak.sql** adatbázist. **Táblát nem kell készíteni: a JPA majd elkészíti.**

src/main/resources/**application.properties**

spring.jpa.hibernate.ddl-auto=update

spring.datasource.url=jdbc:mysql://\${MYSQL\_HOST:localhost}:3306/restgyak

spring.datasource.username=root

spring.datasource.password=

spring.datasource.driver-class-name=com.mysql.jdbc.Driver

**Készítsük el az @Entity Modelt (Java osztály):**

src/main/java/com/example/gyakorlat/**Szemely.java**

**Ez az osztály lesz kapcsolatban az adatbázisban lévő személyek táblával.**

package com.example.gyakorlat;

import javax.persistence.\*;

@Entity

@Table(name="szemelyek")

public class Szemely {

    @Id

    @GeneratedValue(strategy=GenerationType.IDENTITY)      // AUTO\_INCREMENT

    private Long id;

    private String nev;

    private String cim;

    private int kor;

    private double suly;

```
}
```

Generáltassuk le a **getter** és **setter** metódusokat.

Készíttessünk egy paraméter nélküli és a egy 4 paraméteres (név, cím, kor, súly) konstruktort.

**Code menü/Generate**

Készítsünk egy **Repository**-t az személyek tábla egyszerű eléréséhez: **SzemelyRepository** interface.

**Repository-val már többször találkoztunk.**

Ennek segítségével könnyen tudjuk majd elvégezni a következő műveleteket:

- Create new **Szemely**
- Update existing ones
- Delete **Szemely**
- Find **Szemelyek** => Read

src/main/java/com/example/gyakorlat/**SzemelyRepository.java**

```
package com.example.gyakorlat;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
interface SzemelyRepository extends CrudRepository<Szemely, Long> {
}
```

**Futtassuk az alkalmazást.**

**Elkészíti az adatbázisban az üres személyek táblát.**

Készítsünk egy src/main/java/com/example/gyakorlat/**LoadDatabase** osztályt a kezdeti adatok feltöltéséhez az adatbázisba.

```
package com.example.gyakorlat;
```

```
import org.slf4j.Logger;
```

```
import org.slf4j.LoggerFactory;
```

```
import org.springframework.boot.CommandLineRunner;
```

```
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
```

```
@Configuration
```

```
class LoadDatabase {
```

```
 private static final Logger log = LoggerFactory.getLogger(LoadDatabase.class);
```

```
 @Bean
```

```
 CommandLineRunner initDatabase(SzemelyRepository repository) {
```

```
 return args -> {
```

```
 if(true){ // Első feltöltés után állítsuk false-ra, mert minden futtatáskor újra feltöltené
```

```
 repository.save(new Szemely("Kovacs Tibor", "Kecskemet", 35, 77.5));
```

```
 repository.save(new Szemely("Nagy Ilona", "Szeged", 22, 72.3));
```

```
 }
```

```
 };
```

```
 }
```

```
}
```

Futtassuk az alkalmazást => feltölti a táblát az adatokkal

**if(false){ // Első feltöltés után állítsuk false-ra , mert minden futtatáskor újra feltöltené**

**Készítsük el az SzemelyController kontrollert:**

```
package com.example.gyakorlat;
```

```
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;
```

@RestController

```
class SzemelyController {
 private final SzemelyRepository repo;
 SzemelyController(SzemelyRepository repo) {
 this.repo = repo;
 }
 @GetMapping("/szemelyek")
 Iterable<Szemely> olvasMind() {
 return repo.findAll();
 }

 @GetMapping("/szemelyek/{id}")
 Szemely olvasEgy(@PathVariable Long id) {
 return repo.findById(id)
 .orElseThrow(() -> new SzemelyNotFoundException(id));
 }

 @PostMapping("/szemelyek")
 Szemely szemelyFeltolt(@RequestBody Szemely ujSzemely) {
 return repo.save(ujSzemely);
 }

 @PutMapping("/szemelyek/{id}")
 Szemely szemelyModosit(@RequestBody Szemely adatSzemely, @PathVariable Long id) {
 return repo.findById(id)
 .map(a -> {
 a.setNev(adatSzemely.getNev());
 a.setCim(adatSzemely.getCim());
 a.setKor(adatSzemely.getKor());
 a.setSuly(adatSzemely.getSuly());
 return repo.save(a);
 })
 .orElseGet(() -> {
 adatSzemely.setId(id);
 return repo.save(adatSzemely);
 });
 }
 @DeleteMapping("/szemelyek/{id}")
 void torolSzemely(@PathVariable Long id) {
 repo.deleteById(id);
 }
}
```

Készítsük el az **SzemelyNotFoundException** osztályt:

src/main/java/com/example/gyakorlat/**SzemelyNotFoundException.java**

```
package com.example.gyakorlat;
class SzemelyNotFoundException extends RuntimeException {
 SzemelyNotFoundException(Long id) {
 super("A személy nem található: " + id);
 }
}
```

## Az alkalmazás tesztelése cURL -el

Nyissunk egy Parancssort a Windowsban bármelyik mappában

```
curl -h Help
```

### GET

Írjuk be a parancssorba:

```
curl localhost:8080/szemelyek
```

Eredménye:

```
[{"id":1,"nev":"Kovacs Tibor","cim":"Kecskemet","kor":35,"suly":77.5},{ "id":2,"nev":"Nagy Ilona","cim":"Szeged","kor":22,"suly":72.3}]
```

Ezt a mapping-et hívja meg: `@GetMapping("/szemelyek")`

Erre részletesebb kiíratást ad (próbáljuk ki):

```
curl -v localhost:8080/szemelyek
```

Próbáljuk ki egy létező ID-re:

```
curl localhost:8080/szemelyek/1
```

Ezt a mapping-et hívja meg: `@GetMapping("/szemelyek/{id}")`

Próbáljuk ki egy nem létező ID-re:

```
curl localhost:8080/szemelyek/99
```

=> Hibaüzenet

### POST

Egy új Szemelyek rekord felvételéhez a következő POST utasítást használjuk:

```
curl -X POST localhost:8080/szemelyek -H "Content-type:application/json" -d "{\"nev\": \"Kiss Tibor\", \"cim\": \"Szolnok\", \"kor\": \"22\", \"suly\": \"75.6\"}"
```

Nézzük meg az eredményt az adatbázisban.

Ezt a mapping-et hívja meg: `@PostMapping("/szemelyek")`

### PUT

Módosítsuk az ID=3 Szemely kor adatát 27-re

```
curl -X PUT localhost:8080/szemelyek/3 -H "Content-type:application/json" -d "{\"nev\": \"Kiss Tibor\", \"cim\": \"Szolnok\", \"kor\": \"27\", \"suly\": \"75.6\"}"
```

Nézzük meg az eredményt az adatbázisban.

Ezt a mapping-et hívja meg: `@PutMapping("/szemelyek/{id}")`

## **DELETE**

**Töröljük az ID=3 Szemely-t:**

`curl -X DELETE localhost:8080/szemelyek/3`

Nézzük meg az eredményt az adatbázisban.

Ezt a mapping-et hívja meg: `@DeleteMapping("/szemelyek/{id}")`

## Az alkalmazás tesztelése Postman -el

<https://www.postman.com/>

<https://www.postman.com/downloads/>

Portable (nem kell telepíteni):

<https://portapps.io/app/postman-portable/>

Online:

<https://www.postman.com/downloads/>

**Töltsük le a Portable verziót.**

Indítsuk el az exe fájlt => Kicsomagolja a kért mappába.

Indítsuk el az alkalmazást.

Nem kell account-ot regisztrálni.

<https://learning.postman.com/docs/getting-started/introduction/>

<https://learning.postman.com/docs/getting-started/sending-the-first-request/>

<https://learning.postman.com/docs/sending-requests/requests/>

## Create a request

Scratch Pad / New

=> HTTP Request

Scratch Pad

New

## **GET**

`localhost:8080/szemelyek`

GET

localhost:8080/szemelyek

Send

=> Send

**megadja JSON formában az eredményt:**

**Pretty:**

```
[
 {
 "id": 1,
 "nev": "Kovacs Tibor",
 "cim": "Kecskemet",
 "kor": 35,
```

```

 "suly": 77.5
 },
 {
 "id": 2,
 "nev": "Nagy Ilona",
 "cim": "Szeged",
 "kor": 22,
 "suly": 72.3
 }
]

```

**Raw:** [{"id":1,"nev":"Kovacs Tibor","cim":"Kecskemet","kor":35,"suly":77.5},{ "id":2,"nev":"Nagy Ilona","cim":"Szeged","kor":22,"suly":72.3}]

**Próbáljuk ki:**

localhost:8080/szemelyek/1

GET	localhost:8080/szemelyek/1
-----	----------------------------

localhost:8080/szemelyek/99

## **POST**

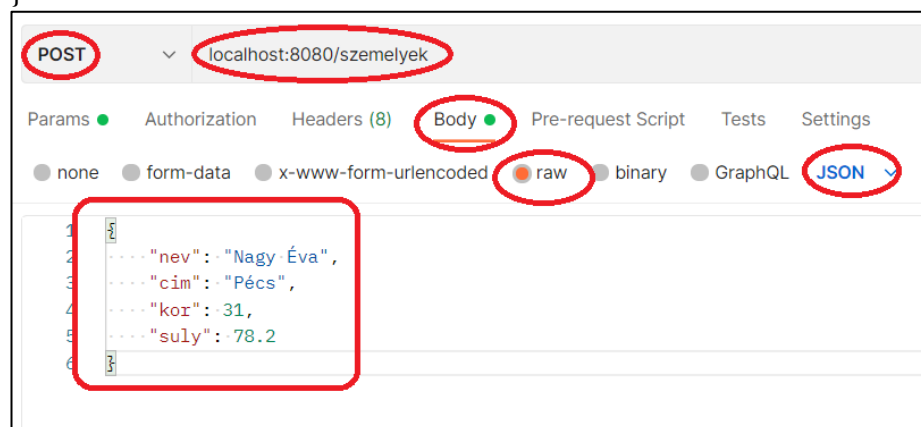
<https://www.toolsqa.com/postman/post-request-in-postman/>

<https://www.javatpoint.com/post-request-in-postman>

```

{
 "nev": "Nagy Éva",
 "cim": "Pécs",
 "kor": 31,
 "suly": 78.2
}

```



=> Send

Nézzük meg az eredményt az adatbázisban.

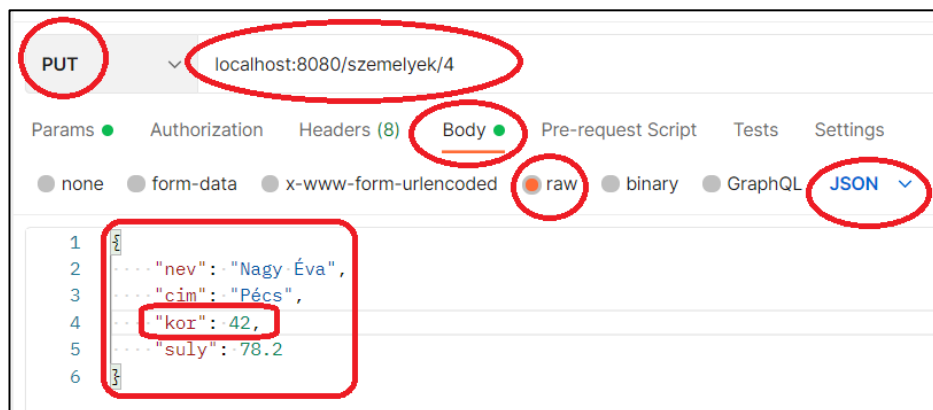
## **PUT**

<http://makeseleniumeasy.com/2019/01/20/api-testing-tutorial-part-20-sending-put-request-in-postman/>

<https://www.educative.io/edpresso/rest-api-test-using-postman>

Előtte nézzük meg a módosítandó rekord ID-jét.

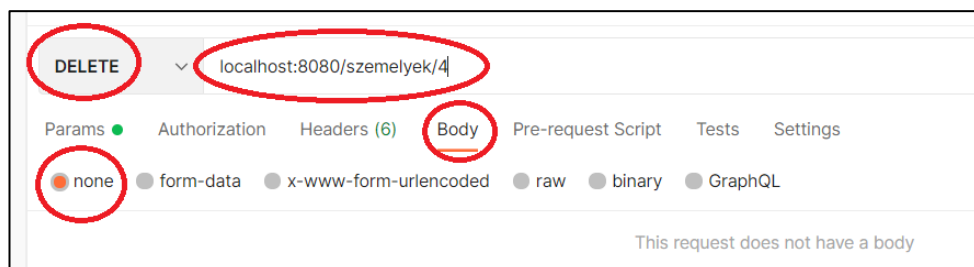




## DELETE

<http://makeseleniumeasy.com/2019/03/04/api-testing-tutorial-part-23-sending-delete-request-in-postman/>

Előtte nézzük meg a törölendő rekord ID-jét.



Nézzük meg az eredményt az adatbázisban.

## Gyakorló házi feladat – Kiegészítés kliens oldali alkalmazással

Az eddigi ismeretekkel meg tudják csinálni.

Egészítsék ki az alkalmazást egy kliens oldali oldallal:

<https://localhost:8080/szemelyek/urlap>

### Beszúrás / Módosítás / Törlés

Id:

Név:  Cím:

Kor:  Súly:

Készítsék el a fenti oldalt az űrlappal.

#### 1. GET

Az oldal minden meghívásakor az űrlap elé írja ki a személyek tábla adatait egy HTMLtáblázatban. Majd az űrlapot is megjeleníteni.

## **2. POST**

Ha a felhasználó nem tölti ki az Id mezőt, de kitölti a többi 4 mezőt: beírjuk az új rekordot a táblába.

## **3. PUT**

Ha a felhasználó kitölti az Id mezőt, és kitölt még legalább egy mezőt: Módosítsuk az adott ID-jű felhasználónak az adott datát, vagy adatait.

## **4. DELETE**

Ha a felhasználó kitölti az Id mezőt, és más mezőt nem tölt ki: Töröljük ki az adott ID-jű rekordot a táblából.

**Az űrlap adatainak elküldése után újra:**

Az űrlap elé írja ki a személyek tábla adatait egy HTMLtáblázatban, majd az űrlapot is megjeleníteni.

## **12. gyakorlat: 2.ZH a Spring anyagából**