Explications liées au programme DemoTest7SGBDH2

<dependency> <groupId>com.h2database</groupId> <artifactId>h2</artifactId> <scope>runtime</scope> </dependency> **<dependency> <groupId> org.springframework.boot </groupId> <artifactId> spring-boot-starter-data-jpa </artifactId> </dependency>** **<dependency>** **<groupId>com.fasterxml.jackson.core</groupId> <artifactId>jackson-databind</artifactId> </dependency>**	Dépendance pour pouvoir utiliser H2 qui est une base de données qui se charge en MC et qui est réinitialisée par défaut à chaque démarrage → utile en phase de développement et de test d'une application, api REST, … qui doivent utiliser un SGBDR Dépendance API JPA Dépendance API Jackson (JSON) → d'office les échanges d'objets entre un Web service REST et un client se fera au format Json par défaut

spring.datasource.url=jdbc:h2:mem:testdb	Fichier **application.properties** : Permet de définir le nom de la BD qui sera créé au démarrage de l'application

insert into User (login, name, bareme) values ('TSE','Severs', 2500), ('DUP','Dupont',3500), ('BER','Bertand',4500)	Les instructions SQL stockées dans un fichier nommé **data.sql** (dans source/main/ressources) seront exécutées au démarrage de l'application → Créer une BD H2 non vide

@Entity @Data @NoArgsConstructor @AllArgsConstructor public class User { @Id private String login; private String name; private float bareme; }	Entity bean User permettant à JPA d'interagir avec la table User dans la BD H2 testdb

public interface UserBD extends **CrudRepository<User,String>** **{** **// User --> classe Entity gérée par UserBD** **// String le type de la clé primaire de User** **}**	**CrudRepository** est une interface proposant des méthodes permettant de réaliser facilement les opérations CRUD sur une table via JPA. C'est Spring qui fournit l'implémentation de ces méthodes findAll, findById, save, delete, ….

//@Component **@Service** **public class AccessBD** **implements CommandLineRunner {**	Créons une classe gérée par Spring (@Component ou @Service) et qui implémente l'interface CommandLineRunner → implémentation d'une méthode run qui sera exécutée automatiquement au lancement de l'application
@Autowired **private UserBD userBD;**	On définit un bean de type UserBd qui sera instancié automatiquement par Spring (@Autowired). Spring fournit l'implémentation des méthodes de l'interface UserBD
@Override **public void run(String... args) throws Exception {** **// CRUD sur une table !!!!** **// Find ALL** **System.out.println(userBD.findAll().toString());**	La méthode findAll renvoie une liste que l'on transforme en String pour simple affichage
// Find by ID **Optional<User>** **optionalUser=userBD.findById("BER");**	La méthode findById renvoie un Optional<User> (objet qui simplifie la gestion des instances null)
if(optionalUser.isPresent()) **{** **System.out.println(optionalUser.get().toString());**	L'instance recherchée (si elle existe) est affichée et puis effacée de la BD
// Delete by Id **User user=optionalUser.get();** **userBD.delete(user);** **}**	On extrait le User présent dans Optional et on l'efface
System.out.println(userBD.findAll().toString());	
// Create (Insert) **user=new User("VAN","VanCau", 2500);** **userBD.save(user);**	On crée un nouvel User que l'on insère dans la BD

System.out.println(userBD.findAll().toString()); **// Update** **optionalUser=userBD.findById("TSE");** **if(optionalUser.isPresent())** **{user=optionalUser.get();** **user.setBareme(3999);** **userBD.save(user);** **}** **System.out.println(userBD.findAll().toString());** **Iterator<User> list=userBD.findAll().iterator();** **while(list.hasNext())** **{User user2=list.next();** **// Trt qcq sur le User** **System.out.println(user2.toString());** **}** **Iterable<User> list2=userBD.findAll();** **for(User user3:list2)** **System.out.println(user3);** **}** **}**	On modifie un User existant (changement de barème) On sauvegarde l'objet modifié. C'est Spring qui gère la différence entre un Insert et un Update au niveau de la méthode save() Les objets User renvoyés par findAll doivent pouvoir être traités séparément. On les récupère via un iterator Idem, mais on utilise l'objet Iterable<User> renvoyé directement par findAll → utilisation d'une boucle « foreach »

On peut également accéder à la BD via des Rest Web Services

@RestController **@RequestMapping("api")** **public class ServiceRest {** **@Autowired** **UserBD userBD;**	
@GetMapping("find") **public String find(@RequestParam String login)** **{ Optional<User> opt=userBD.findById(login);** **if(!opt.isPresent()) return "Pas trouvé";** **return opt.get().toString();** **}**	L'objet trouvé est renvoyé sous forme de String juste pour test
@GetMapping("findjson") **public User findJson(@RequestParam String login)** **{ return userBD.findById(login).get();** **}**	L'objet est renvoyé au format Json par Spring car on a mis une dépendance à Jackson
@GetMapping("findall") **public Iterable<User> findAll()** **{return userBD.findAll();}**	La liste d'objets est renvoyée au format json. Attention, on sait récupérer cette liste et l'afficher via un navigateur, Postman, Curl. Mais avec un client restTemplate on ne sait pas récupérer les différents Employes séparément. Utiliser un tableau ou encapsuler la liste dans une classe (voir cours précédent)
@GetMapping("save") **public String save(@RequestParam String login, @RequestParam String name, @RequestParam String bareme)** **{** **User user =new User(login, name, Float.valueOf(bareme));** **userBD.save(user);** **return user.toString()+ "a été ajouté";** **}**	On aurait pu utiliser une requête Post initialisée par un formulaire

```
@GetMapping("delete")
    public String delete(@RequestParam String login)
  {
     Optional<User> opt=userBD.findById(login);
     if(!opt.isPresent()) return "Pas trouvé";
     User user=opt.get();
     userBD.delete(user);
     return user.toString()+ "a été effacé";

  }



@GetMapping("update")
  public String update(@RequestParam String login,
  @RequestParam String bareme)
  {
     Optional<User> opt=userBD.findById(login);
     if(!opt.isPresent()) return "Pas trouvé";
     User user=opt.get();
     user.setBareme(Float.valueOf(bareme));
     userBD.save(user);
     return user.toString()+ "a été modifié";

  }


}
```