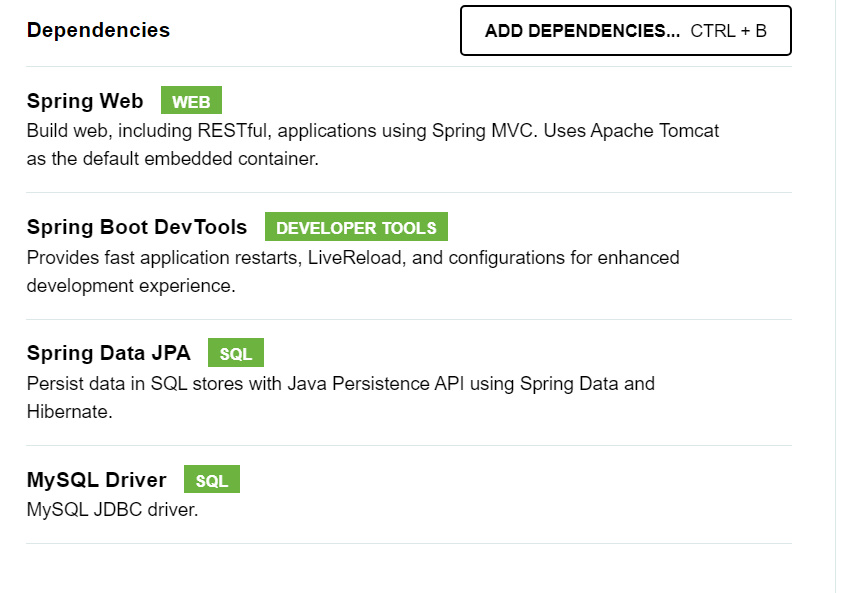
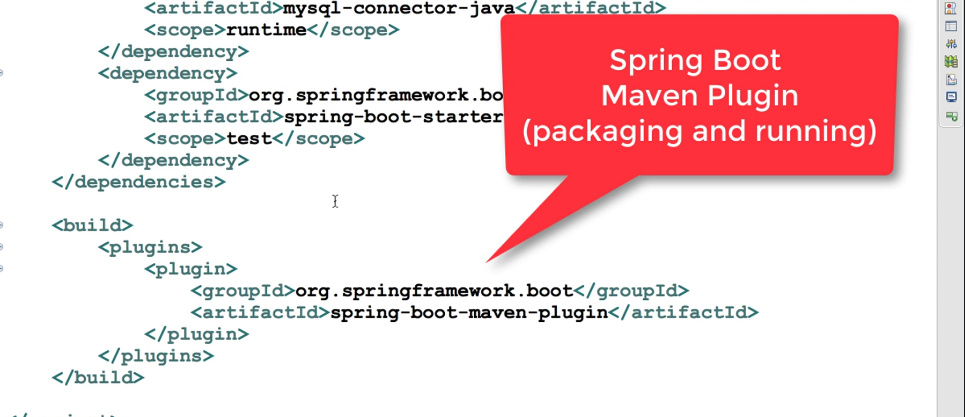
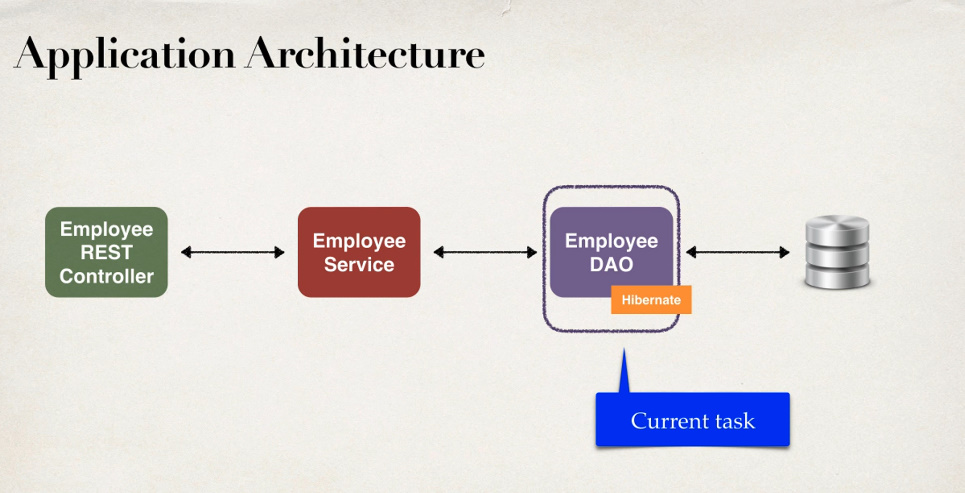
**ATENTIE! Spring 6 nu suporta JavaX, ci doar Jakarta!!!!**



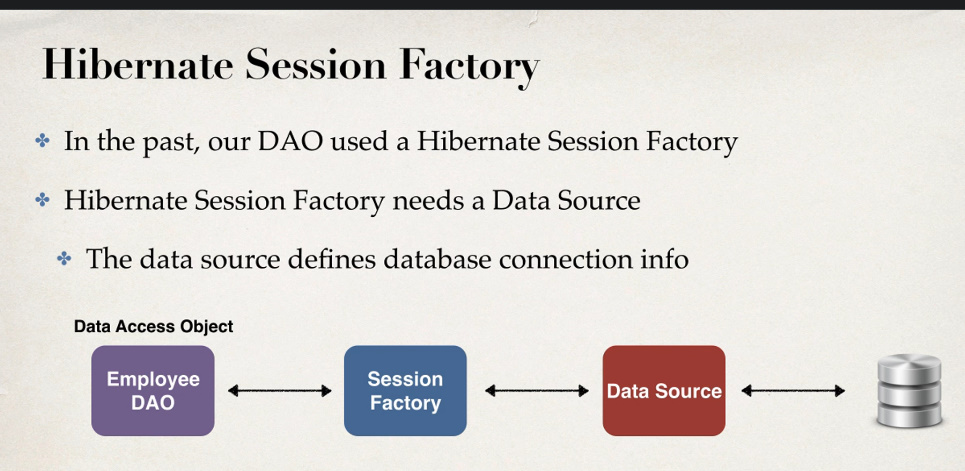


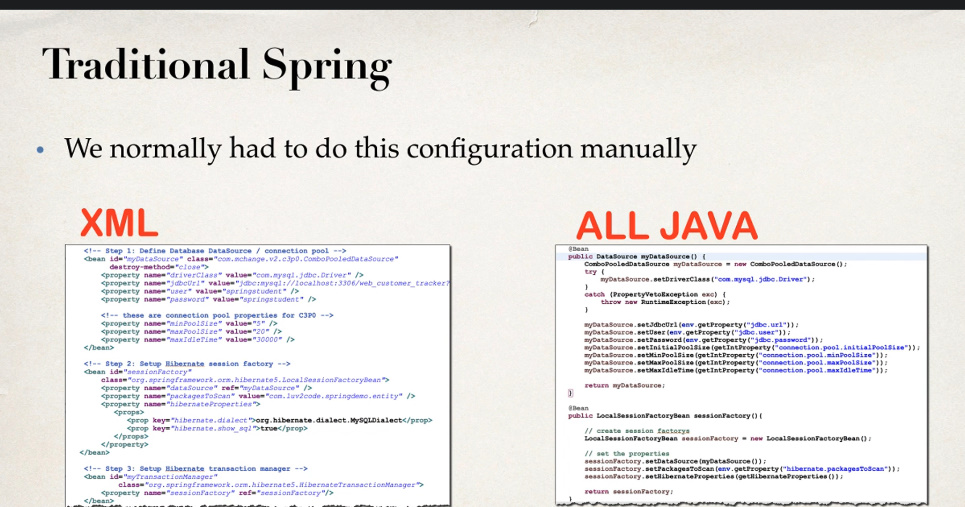


**App Architecture**

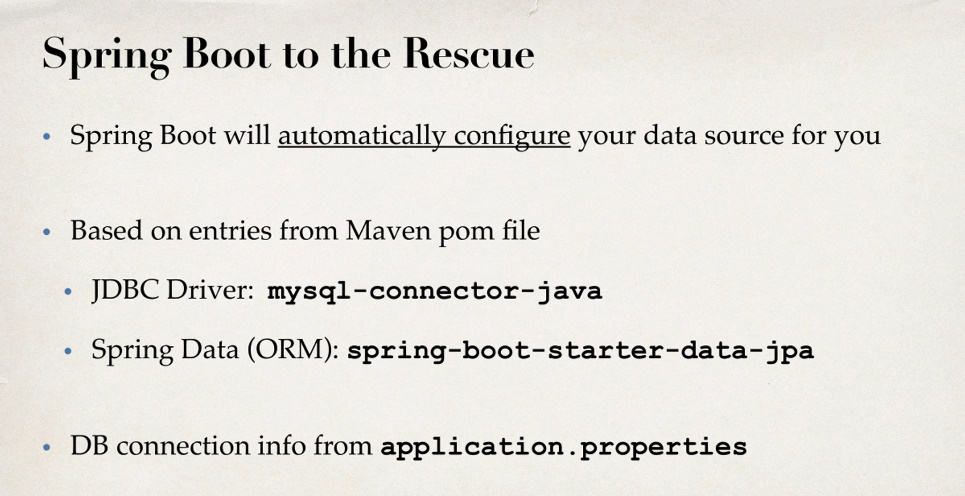


**Integrate Hibernate and JPA**

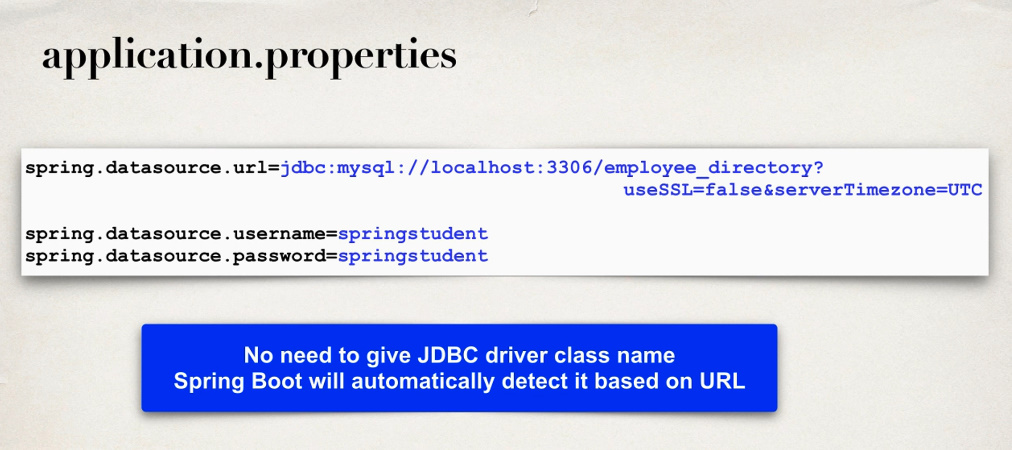




Dar exista o solutie mai usoara pentru a face asta:

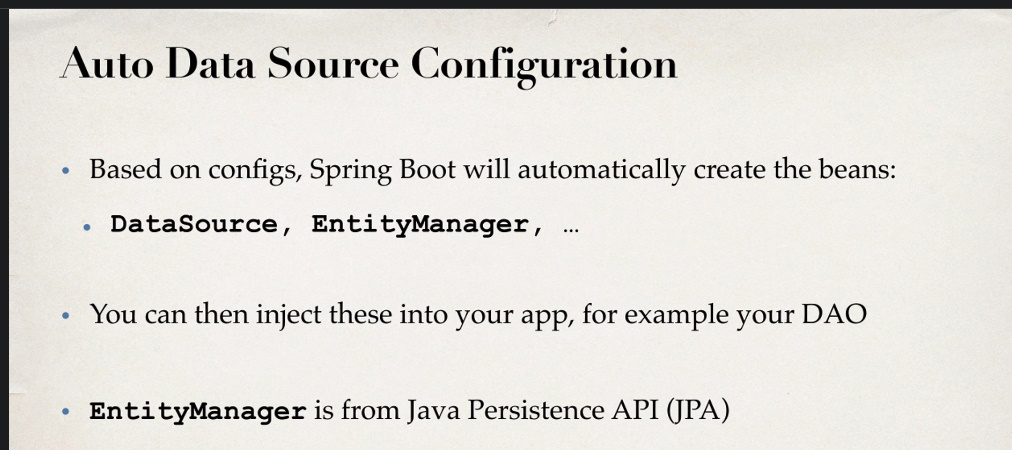
* 

Spring Boot face totul mai usor, si trebuie doar sa setam totul in application.properties

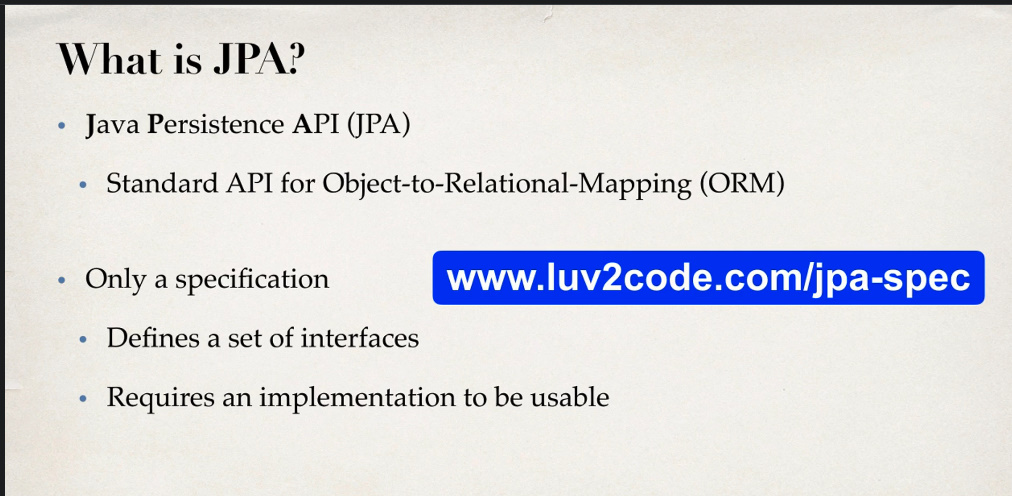
* 

Nu avem nevoie de JDBC Driver class, asa cum din URL e clar ce folosim, mysql sau alta

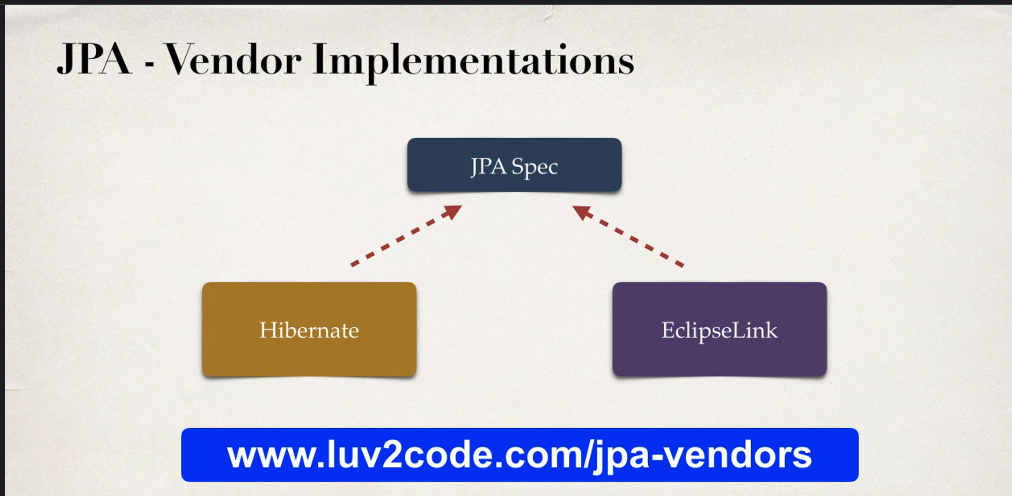
**Mare grija la &serverTimezone=UTC**

* 

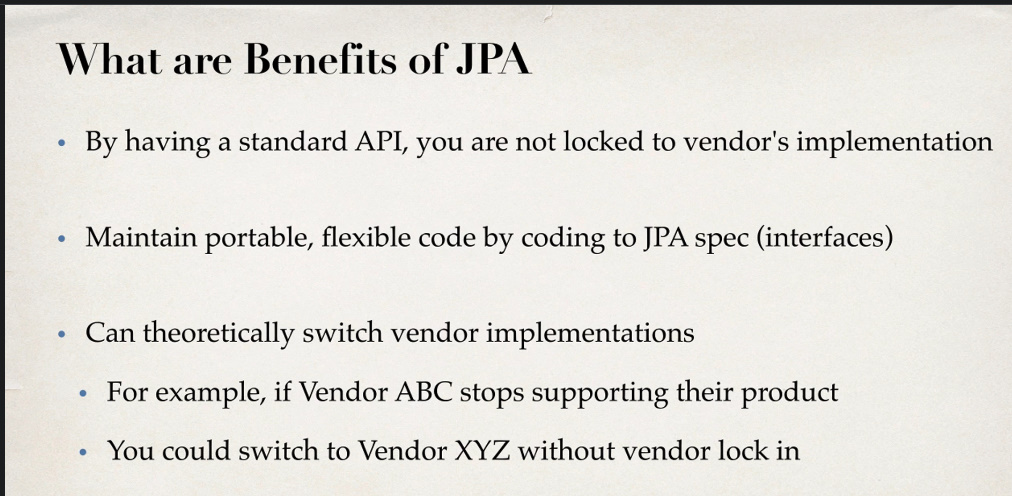
**JPA**

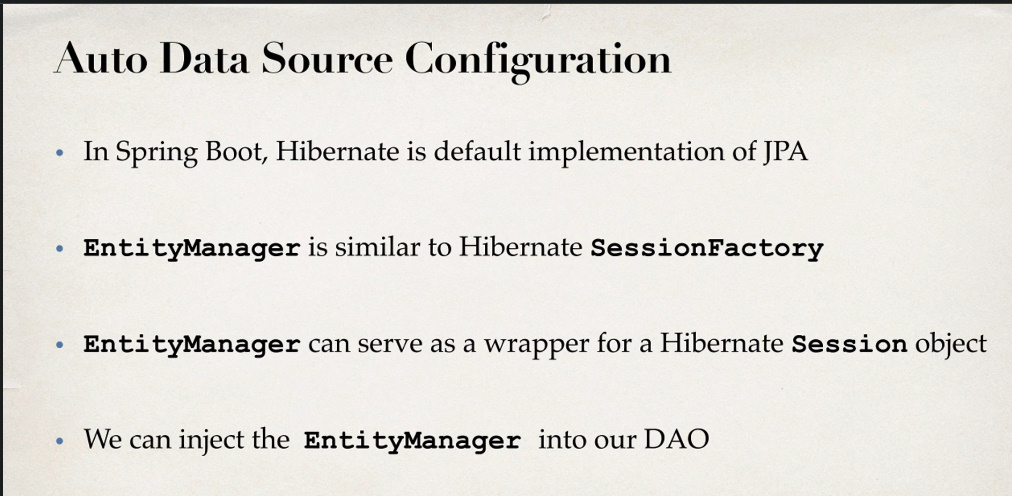


JPA este o colectie de interfete ce trebuie implementate pentru a putea sa fie utilizate.



* Un mare avantaj al JPA este ca Hibernate si alte vendors il implementeaza, si daca gen nu ne place Hibernate, putem trece la alt vendor, ca EclipseLink sau spring Data si nu trebuie sa modificam codul nostru, asa cum ele oricum implementeaza interfetele din JPA, deci vom modifica doar vendorul si atat.



* 

Hibernate implementeaza JPA.

EntityManager e similar cu Session, iar SessionFactory defapt si implementeaza EntityManagerFactory,care contine metode ce functioneaza cu EntityManager, si in plus EntityManager e doar o interfata de la JPA ce necesita implementare, si anume SessionFactory implementeaza EntityManagerFactory

Deci:

SessionFactory implements EntityManagerFctory

Session implements EntityManager

public interface EntityManagerFactory extends AutoCloseable {  
 EntityManager createEntityManager();  
  
 EntityManager createEntityManager(Map var1);  
  
 EntityManager createEntityManager(SynchronizationType var1);  
  
 EntityManager createEntityManager(SynchronizationType var1, Map var2);  
  
 CriteriaBuilder getCriteriaBuilder();  
  
 Metamodel getMetamodel();  
  
 boolean isOpen();

........

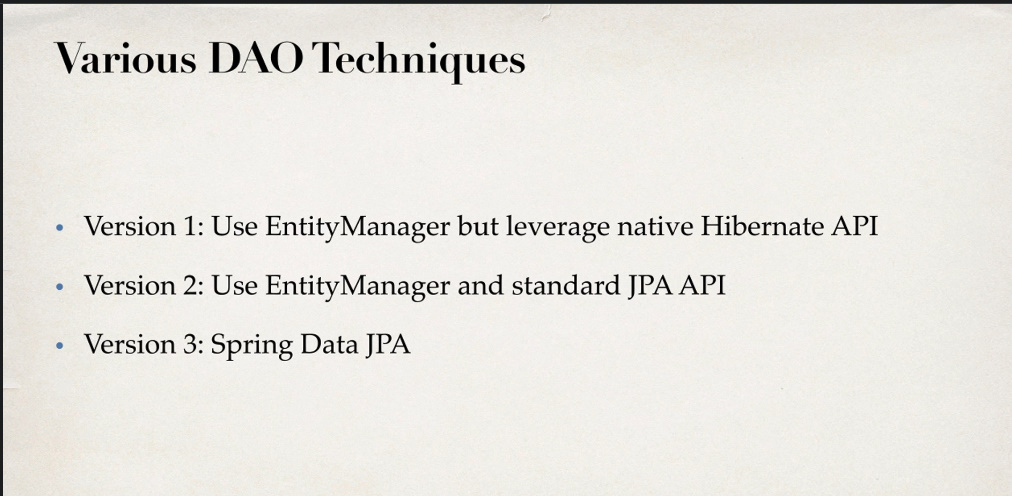
public interface Session extends SharedSessionContract, EntityManager {  
 void flush();

......

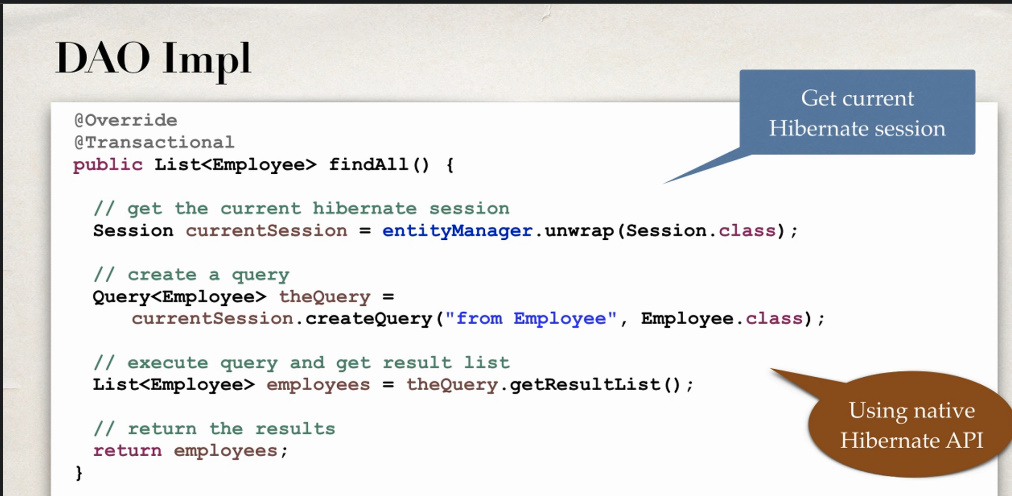
public interface SessionFactory extends EntityManagerFactory, Referenceable, Serializable, Closeable {  
 SessionFactoryOptions getSessionFactoryOptions();  
  
 SessionBuilder withOptions();

.....

Atentie, **SessionFactory** e o interfata! **LocalSessionfactory** nu o implementeaza! dar o foloseste

* 

**Hibernate API**

****

**Nu uitam ca nu folosim javax.persistence ci Jakarta!!!**

* @Repository  
  public class EmployeeDAOImpl implements EmployeeDAO{  
    
   private EntityManager entityManager;  
    
   @Autowired  
   public EmployeeDAOImpl(EntityManager entityManager) {  
   this.entityManager = entityManager;  
   }  
    
   @Override  
   @Transactional  
   public List<Employee> findAll() {  
    
   Session session = entityManager.unwrap(Session.class);  
    
   Query<Employee> query = session.createQuery("from Employee",Employee.class);  
   List<Employee> employees = query.getResultList();  
    
   return employees;  
   }

Session e implementarea lui EntityManager:

public interface Session extends SharedSessionContract, EntityManager {

* Deci, vom avea nevoie de un bean de tip EntityManager
* Acest EntityManager e ca un Session. Metoda folosita pentru a returna o sesiune este unwrap(Session.class)

**C3P0 support**

Daca vrem sa lucram cu C3P0 in properties, vom folosi:

spring.jpa.properties.hibernate.c3p0.timeout = 3000  
spring.jpa.properties.hibernate.c3p0.idle\_test\_period = 2000  
spring.jpa.properties.hibernate.c3p0.min\_size = 5  
spring.jpa.properties.hibernate.c3p0.max\_size = 20

timeout – perioada de timp dupa care, daca o sesiune nu mai e utilizata, va fi stearsa automat

idle\_test\_period – perioada de timp peste care se verifica conexiunile pentru a vedea daca mai sunt valide.

**Atentie, idle nu poate depasi timeout!!!!**

**Vechea Metoda**

* Putem face totul si dupa vechea metoda, cu bean pentru datasource, localsessionfactory si hibernatetransactionmanager.
* Trebuie doar sa punem beanurile in clasa de baza a proiectului si sa mai includem dependentele pentru c3p0
* Atentie!!!

@SpringBootApplication(exclude = HibernateJpaAutoConfiguration.class)

e nevoie sa excludem configuratia automata pentru JPA. De altfel, se va incurca cu de la Hibernate.

* Atentie la asta:
* @Bean(name="entityManagerFactory")  
  public LocalSessionFactoryBean sessionFactory(){

SprigBoot are mereu nevoie de un bean cu asa nume, daca nu folosim exclude = HibernateJpaAutoConfiguration.class

@SpringBootApplication(exclude = HibernateJpaAutoConfiguration.class)  
@EnableTransactionManagement  
public class DemoApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(DemoApplication.class, args);  
 }  
  
 @Bean  
 public DataSource dataSource(){  
 ComboPooledDataSource dataSource = new ComboPooledDataSource();  
 dataSource.setMinPoolSize(5);  
 dataSource.setMaxIdleTime(3000);  
 dataSource.setInitialPoolSize(5);  
 try {  
 dataSource.setLoginTimeout(5000);  
 } catch (SQLException e) {  
 throw new RuntimeException(e);  
 }  
 try {  
 dataSource.setDriverClass("com.mysql.jdbc.Driver");  
 } catch (PropertyVetoException e) {  
 throw new RuntimeException(e);  
 }  
  
 dataSource.setJdbcUrl("jdbc:mysql://localhost:3306/employee\_directory?useSSL=false&serverTimezone=UTC");  
 dataSource.setUser("testuser");  
 dataSource.setPassword("Frb2eshox!");  
  
 return dataSource;  
 }  
  
 @Bean(name="entityManagerFactory")  
 public LocalSessionFactoryBean sessionFactory(){  
 LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();  
 sessionFactoryBean.setPackagesToScan("com.app.demo");  
 sessionFactoryBean.setDataSource(dataSource());  
  
 return sessionFactoryBean;  
 }  
  
 @Bean  
 @Autowired  
 public HibernateTransactionManager transactionManager(SessionFactory sessionFactory){  
 HibernateTransactionManager transactionManager = new HibernateTransactionManager();  
  
 transactionManager.setSessionFactory(sessionFactory);  
  
 return transactionManager;  
 }  
  
  
}

Vezi in projects

**DataSource**

* Asa cum Spring Boot creaza automat beanurile pentru EntityManager si TransactionManager, putem defini pur si simplu un bean de tip DataSource si gata, de celelalte 2 se va ocupa hibernate:

@Bean  
DataSource dataSource(){  
 DataSourceBuilder dataSourceBuilder = DataSourceBuilder.*create*();  
 dataSourceBuilder.driverClassName("com.mysql.jdbc.Driver");  
 dataSourceBuilder.url("jdbc:mysql://localhost:3306/mysqltutorial?useSSL=false&serverTimezone=UTC");  
 dataSourceBuilder.username("testuser");  
 dataSourceBuilder.password("Frb2eshox!");  
  
 return dataSourceBuilder.build();  
}

sau

@Bean("dataSource2")  
public DataSource dataSource(){  
 DriverManagerDataSource dataSource = new DriverManagerDataSource();  
 dataSource.setUsername("testuser");  
 dataSource.setPassword("Frb2eshox!");  
 dataSource.setUrl("jdbc:mysql://localhost:3306?test?useSSL=false");  
 dataSource.setDriverClassName(com.mysql.jdbc.Driver.class.getName());  
   
 return dataSource;  
}

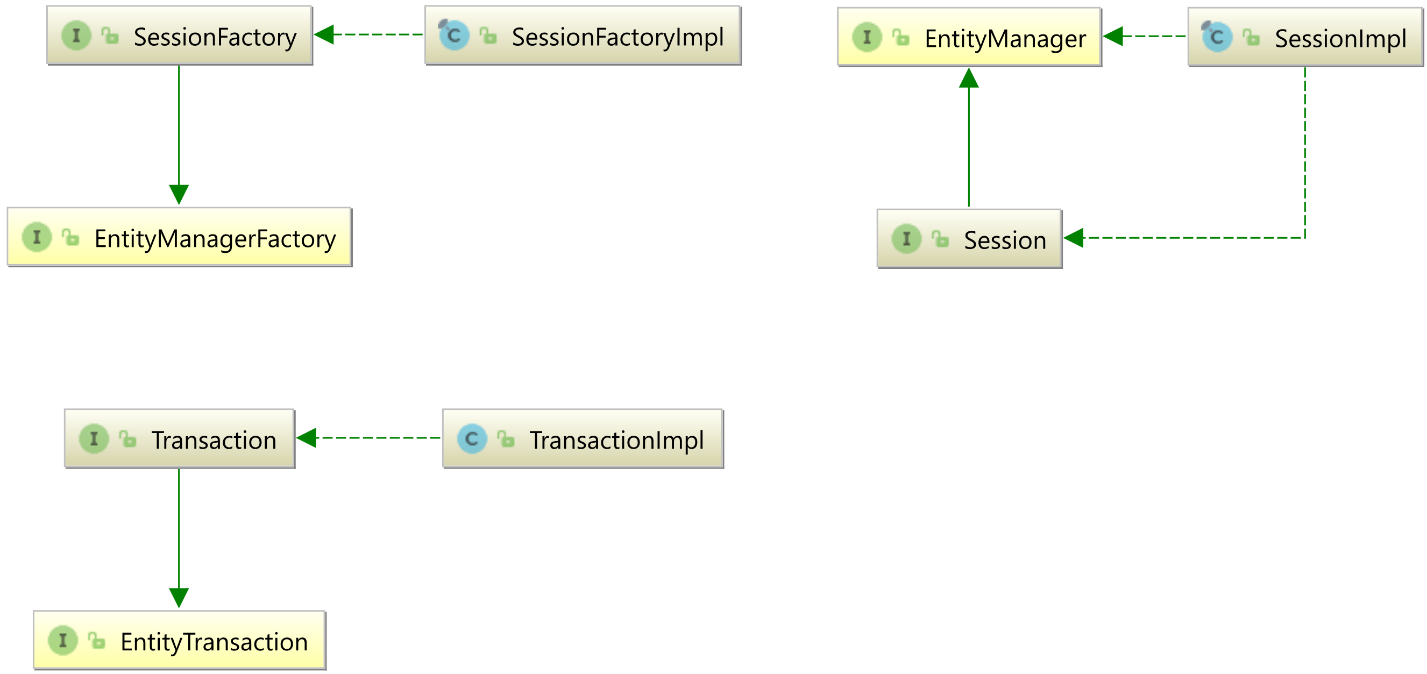
**EntityManagerFactory**

* Putem crea si un entityManagerFactory:

@Bean  
LocalContainerEntityManagerFactoryBean entityManagerFactory(){  
 LocalContainerEntityManagerFactoryBean factoryBean = new LocalContainerEntityManagerFactoryBean();  
 factoryBean.setDataSource(dataSource());  
 factoryBean.setPackagesToScan("com.luv2code.springmvc");  
  
 JpaVendorAdapter jpaVendorAdapter = new HibernateJpaVendorAdapter();  
 factoryBean.setJpaVendorAdapter(jpaVendorAdapter);  
  
 return factoryBean;  
}

**EntityManager vs SessonFactory**

* Deci, desi Spring Data foloseste in Repository create de el un bean de tip EntityManager, putem liber sa cream un SessionFactory pentru el.



* SessionFactory implementeaza EntityManagerFactory
* SessionFactory implementeaza Session,iar Session implementeaza EntityManager iata de ce e posibil asa ceva:
* EntityManager sessionFactoryBean = new LocalSessionFactoryBean().getObject().getCurrentSession();
* Dar e posibil si asa ceva:

EntityManagerFactory sessionFactoryBean = new LocalSessionFactoryBean().getObject();

* In Mod normal, Spring Data foloseste in repository creat un EntityManager,

dar nu e nicio problema daca noi cream un bean de tip SessionFactory sau LocalSessionFactoryBean cu numele “entityManagerFatory”, caci or

icum din LocalSessionFactoryBean se poate ajunge la EntityManager

@Bean("entityManagerFactory")  
 public LocalSessionFactoryBean sessionFactory() throws PropertyVetoException {  
LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();  
sessionFactoryBean.setDataSource(dataSource());  
sessionFactoryBean.setPackagesToScan("com.example.test");  
sessionFactoryBean.setHibernateProperties(hibernateProperties());  
  
return sessionFactoryBean;  
  
 }

public interface TeacherDAO extends JpaRepository<Teacher,Integer> {  
}

**Si iata cum se ajunge:**

LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();  
EntityManagerFactory entityManagerFactory = sessionFactoryBean.getObject();  
EntityManager entityManager = entityManagerFactory.createEntityManager();

**sau si asa:**

LocalSessionFactoryBean sessionFactoryBean = new LocalSessionFactoryBean();  
EntityManager entityManager = sessionFactoryBean.getObject().getCurrentSession();

* **Chiar daca avem un LocalSessionFactoryBean, un Entity poate avea ca AutoWired un EntityManager, si eroare ca beanul nu e egasit e fasla:**

@Repository  
public class TeacherDAO {  
 @Autowired  
 private EntityManager entityManager;

**Ce implementari ale lui EntityManager si EntityManagerSession folosesc Spring Boot/Hibernate?**

* JPA contine doar interfete, deci nu putem propriu zis sa lucram cu JPA
* JPA are nevoie de o implementare, si Hibernate implementeaza toate interfatele sale.
* Deci, este evident ca Hibernate nu creaza beanuri de tip EntityManager sau EntityManagerFactory, caci astea doar sunt interfete
* **Hibernate foloseste SessionFactory ca implementare pentru EntityManagerFactory**
* **Hibernate foloseste Session ca implementare pentru EntityManager**
* Iata de ce, urmatoarele coduri vor fi valide:

@Controller  
public class MyController {  
 @Autowired  
 private EntityManagerFactory entityManagerFactory;  
   
 @PostConstruct  
 public void test(){  
 SessionFactory sessionFactory = (SessionFactory) entityManagerFactory;  
 System.*out*.println(sessionFactory);  
 }  
}

si:

public class MyController {  
 @Autowired  
 private EntityManager entityManager;  
  
 @PostConstruct  
 public void test(){  
 Session sessionFactory = (Session) entityManager;  
 System.*out*.println(sessionFactory);  
 }  
}



**Castul s-a facut cu succes.**

* Spring Boot se asigura sa ofere si un bean ce returneza sesiuni ale lui SessionFactory, adica EntityManagerFactory, iata de ce putem da @Autowired si la EntityManager
* **Atentie! Nu putem da @Autowired la SessionFactory sau Session, deoarece Spring Boot nu face down-casting automat!**