* <dependencyManagement></ dependencyManagement> allows us to set up dependencies that can be inherited from POM downstream.
* When we install the project, project jar file going to save locally in .m2 folder.

(Installing D:\Java\_Learning\Microservices\_Guru\mssc-brewery-bom\pom.xml to C:\Users\User\.m2\repository\guru\springframework\mssc-brewery-bom\0.0.1-SNAPSHOT\mssc-brewery-bom-0.0.1-SNAPSHOT.pom)

1. What is datasource configuration ?

It tells java how to connect JDBC complaint Data source.

JDBC is a standard for connecting to relational databases.

**H2 DATABASE**

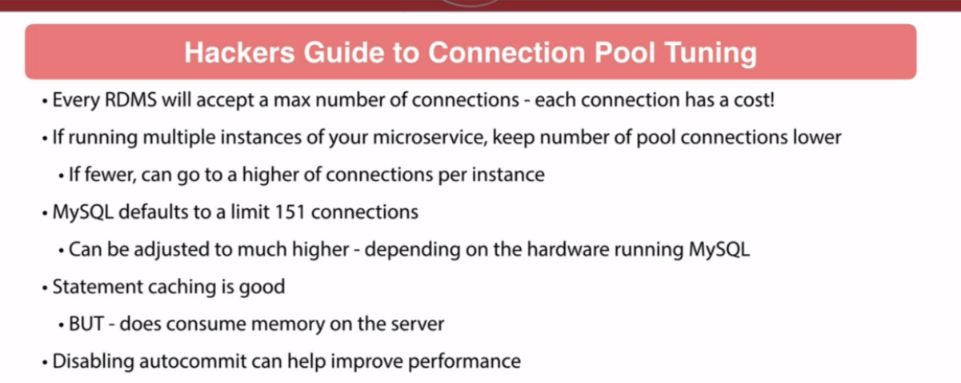
* H2 db preferred for local development
* If spring dev tools is on classpath DB console is auto configured for browsing the db or add “spring.h2.console.enabled=true”.
* H2 has compatibility mode.

“spring.datasource.url = jdbc:h2:mem:testdb;MODE=MYSQL”.

**Bootstrap data from data.sql (Baeldung)**

[**https://www.baeldung.com/spring-boot-data-sql-and-schema-sql**](https://www.baeldung.com/spring-boot-data-sql-and-schema-sql)

**Hikari Connection Pooling**

****

* Messaging feature: Spring for Apache ActiveMQ Artemis is by RedHat

ObjectOptimisticLockingFailureException: Object Updated before saving it.

# Error!!!: [PersistentObjectException: detached entity passed to persist thrown by JPA and Hibernate](https://stackoverflow.com/questions/13370221/persistentobjectexception-detached-entity-passed-to-persist-thrown-by-jpa-and-h)

# Reason1: There are 2 entity A & B. A has @OneToMany and B has @ManyToOne.

# Both With cascadeType.ALL (detach, merge, refresh, persist, delete, remove).

# B has A in its properties, so to save B we must create A. So if we simply do

# A a = new A(); then pass “a” to B (setter). Above error will throw. Bcz in B we have used CascadeType.ALL (which has “persist”) So it is persisting (saving) A also which is already exist in db.

# Solution 1: Remove CascadeType.Persist while mapping.(risky as mentioned below)

# \* Reason 2: transaction is commited and closed immediately after we call save, saveAll…etc.

# So when we try to use the same object again, the above exception will throw.

# \* Solution 2: add @Transactional to the method or class level.

Points: CascadeType info

StackOverflow: https://stackoverflow.com/questions/13370221/persistentobjectexception-detached-entity-passed-to-persist-thrown-by-jpa-and-h

- By saying "cascade ALL" on the child entity Transaction, you require that every DB operation gets propagated to the parent entity Account. If you then do persist(transaction), persist(account) will be invoked as well.

But only transient (new) entities may be passed to persist (Transaction in this case). The detached (or other non-transient state) ones may not (Account in this case, as it's already in DB).

Therefore you get the exception "detached entity passed to persist". The Account entity is meant! Not the Transaction you call persist on.

**-** Using merge is risky and tricky, so it's a dirty workaround in your case. You need to remember at least that when you pass an entity object to merge, it stops being attached to the transaction and instead a new, now-attached entity is returned. This means that if anyone has the old entity object still in their possession, changes to it are silently ignored and thrown away on commit.

You are not showing the complete code here, so I cannot double-check your transaction pattern. One way to get to a situation like this is if you don't have a transaction active when executing the merge and persist. In that case persistence provider is expected to open a new transaction for every JPA operation you perform and immediately commit and close it before the call returns. If this is the case, the merge would be run in a first transaction and then after the merge method returns, the transaction is completed and closed and the returned entity is now detached. The persist below it would then open a second transaction, and trying to refer to an entity that is detached, giving an exception. Always wrap your code inside a transaction unless you know very well what you are doing.

# Observation: How to update already existing object in db. (here A & B has @ManyToMany, mappedby A , also includes @JoinTable(..joincolumn, inverseJoincolumn…) on A only, cascadeType is only MERGE on both A & B)

Authority authoritySuperAdmin = authorityRepository.findById(1L).get();  
User user = userRepository.findByUsername("user2").get();  
  
user.getAuthorities().add(authoritySuperAdmin);  
userRepository.save(user);

Error!!!: StackOverflow error when we try to fetch details when @oneToMany for one entity & @ManyToOne for another entity.

**Reason:** When we override toString() method it will recursivly call each other, resulting in stackoverflow error.

Solution: don’t override toString() method.

**Map Stream Example:**

List<Integer> num = Arrays.asList(1,2,3,4,5);

List<Integer> collect1 = num.stream().map(n -> n \*2 ).collect(Collectors.toList());

System.out.println(collect1); *//[2, 4, 6, 8, 10]*

**Filter Stream Example:**

Stream<String> stream = Stream.of(

            "Geeks", "foR", "GeEksQuiz", "GeeksforGeeks");

        // Getting a stream consisting of the

        // elements ending with 's'

        // using Stream filter(Predicate predicate)

        stream.filter(str -> str.endsWith("s"))

            .forEach(System.out::println);

**UserDetailsService in springboot \*\*\***

**Question**: I'm creating authentication service in Spring.

I'm using UserDetailsService to get form variables, but i found that loadUserByUsername has only one variable - userName.

*@Service*

public class CustomUserDetailService implements UserDetailsService {

*@Autowired*

private AppUserRepository appUserRepository;

*@Override*

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

AppUser appUser = appUserRepository.findByUserName(username).orElseThrow(() -> new ResourceNotFoundException("User not found with username:"+username));

return appUser;

}

}

**Answer**: If you look at the [User](http://static.springsource.org/spring-security/site/docs/3.0.x/apidocs/org/springframework/security/core/userdetails/User.html) object, the second parameter in the constructor is the password.

The [UserDetailsService](http://static.springsource.org/spring-security/site/docs/3.0.x/apidocs/org/springframework/security/core/userdetails/UserDetailsService.html) is used to load the user from a back-end structure like database. The [loadUserByUsername](http://static.springsource.org/spring-security/site/docs/3.0.x/apidocs/org/springframework/security/core/userdetails/UserDetailsService.html#loadUserByUsername%28java.lang.String%29) method is called when a user tries to login with a username and password, then it is the responsibility of the service to load the user definition and return it to the security framework. The required details includes data like username, password, accountNonExpired, credentialsNonExpired, accountNonLocked and authorities.

Once the spring security receives the user object, it will validate the user against the password entered by the user and other data like user account status (accountNonExpired, credentialsNonExpired etc)

Adding remote/other git repo address to our git (for comparision)

>git remote add “name” “repo url”

>git fetch “name”

**Diff between save() and saveAndFlush()**

save()

Normally, Hibernate holds the persistable state in memory. The process of synchronizing this state to the underlying DB is called flushing.

When we use the save() method, **the data associated with the save operation won't be flushed to the DB unless, and until, an explicit call to the flush()** **or commit() method is made**.

If we use JPA implementations like Hibernate, then that specific implementation will be managing the flush and commit operations.

saveAndFlush()

saveAndFlush() method **flushes the data immediately during the execution**

**Note:**

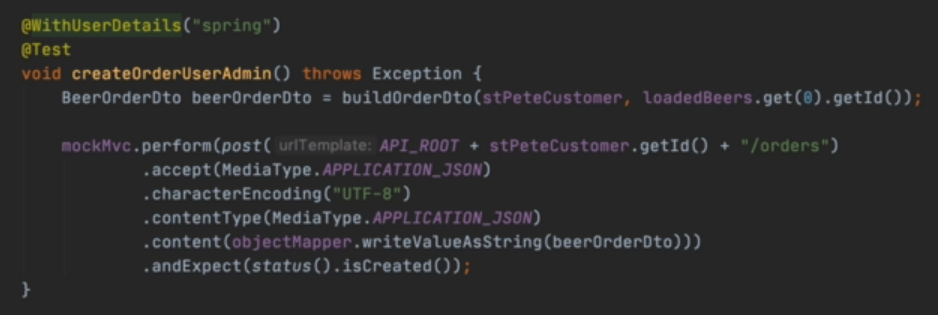
-we can use createdTime and UpdatedTime automatically in a entity.

@CreationTimestamp  
@Column(updatable = false)  
private Timestamp createdDate;  
  
@UpdateTimestamp  
private Timestamp lastModifiedDate;

-Creating pk with UUID

@Id  
@GeneratedValue(generator = "UUID")  
@GenericGenerator(  
 name = "UUID",  
 strategy = "org.hibernate.id.UUIDGenerator"  
)  
@Type(type="org.hibernate.type.UUIDCharType")  
@Column(length = 36, columnDefinition = "varchar", updatable = false, nullable = false )  
private UUID id;

**Note: Instead of configuring the spring username and password in the testcase, we can directly provide userdetails with @WithUserDetails(“username”)­­**



**Note:** Add ModelMapper dependecy to convert model <-> dto.

Ex:

private final ModelMapper modelMapper;

--------

User user = userService.getUser(userId);

return modelMapper.map(user, UserDto.class);

customizing model mapper : <https://craftingjava.com/blog/user-management-implement-rest-api/>

Checkout using tags

>git checkout tags/chapter\_3\_2 -b origin