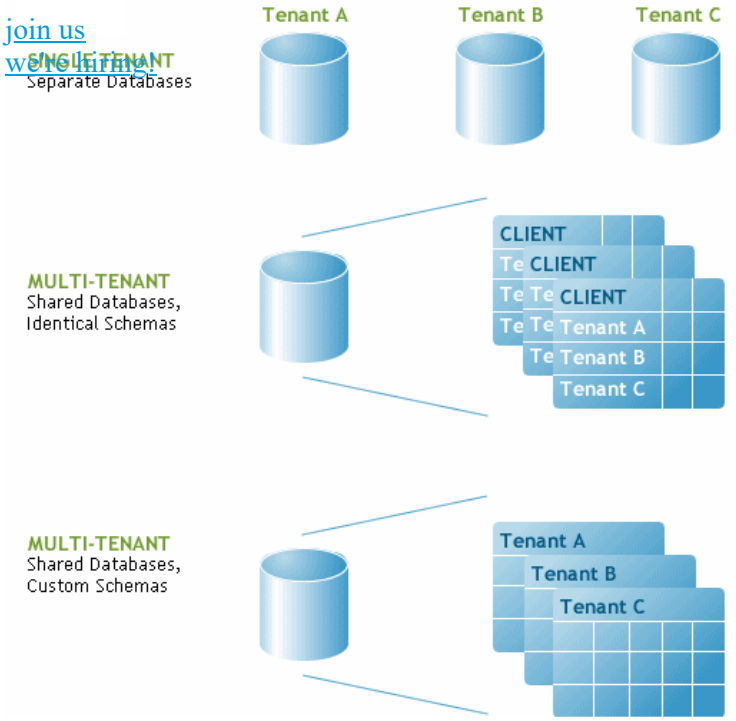
Inject multitenancy in your web application

Arnaud Giuliani | Ekito members | 6 minutes read

Software architecture rework during a project is not always a good news: it induces transverse changes for your application that can become heavy breaking changes.

Multitenancy is the architecture feature that interest us for this article. We can consider the following approaches :

- database level : the app is using distinct database for each customer (isolation at service level)
- data level : the app is using only one database in one server, requests are filtered by discriminant data (isolation at data level)

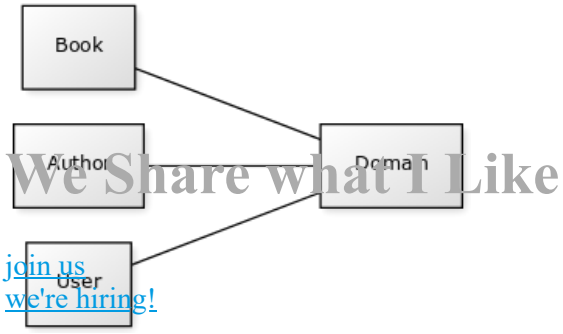


Today, we have a web application that evolves to host all its customers data in the same database schema (before, each customer data was stored in seperated database). A concept of “domain” will help us to make a logical seperation of these customers data (each customer data is associated to a domain). Our goal here: inject the multi-tenancy feature into this existing web application.

Stop talking … let’s go concrete ! Here we have a web application developed with Jhipster generated Spring stack (spring boot, spring data …) and a Mongo database. (Start project is here)

We want a data level isolation multi-tenancy system. We have already dozens of services/repository (Book and Author entities) and the application is evolving to become a multi-tenant application … Ouch ! We need to handle a new “domain” entity. We want to go from this :

t this :



Let's make our Domain entity. We begin with our multi-tenant entity abstraction, that will be inherited by all of our entities classes :

Multitenancy.java

```
public abstract class MultitenantEntity {

    @NotNull
    @DBRef
    Domain userDomain;
```

Let's build our new "databoostrap" and plug it in our main Spring java configuration :

data initialization : DataBootstrap.java

DataBoostrap at startup : Application.java

```
public class Application implements InitializingBean{
...
@Override
public void afterPropertiesSet() throws Exception {
 dataBootstrap.initData();
...
```

Nice, but now we can't do anything on entities (the @NotNull validation annotation doesn't allow us to modify our entity without an associated domain). Ok, What is the situation ?

- Client UI save user details with the new domain reference.
- We don't want to review each line of code : we have the domain data (in the user details), we just have to use it a right time
- But, we have already secured our app and when a user is logged we know every details. It's easy then to keep tracks of its domain.

Adding a new "domain" parameter everywhere to each java method is not smart at all and it would also force us to question about spring data advantages ?! (we couldn't use interface declaration anymore). Are we screwed to come back a lower mongodb programming level ?

But your little voice in your head (sure, you have one) must be screaming : you do the same thing everywhere, it's an architecture aspect … use aspect programming ! 🙂

What is the plan ? We can work with implicit "domain" information everywhere : each repository will use the needed domain parameter, but this one will not be explicitly passed through methods.

At first, we can obtain the user domain from current session :

```
public static Optional<Domain> getCurrentDomain() {
    Domain domain = null;
    //retrieves the current user's login from the security context
    String currentLogin = getCurrentLogin();
    if (currentLogin != null) {
       if (userRepository != null) {
           User user = userRepository.findOne(currentLogin);
           domain = user.getUserDomain();
           //...
```

SecurityUtils.java

Now what we can do with the needed repository operations ?

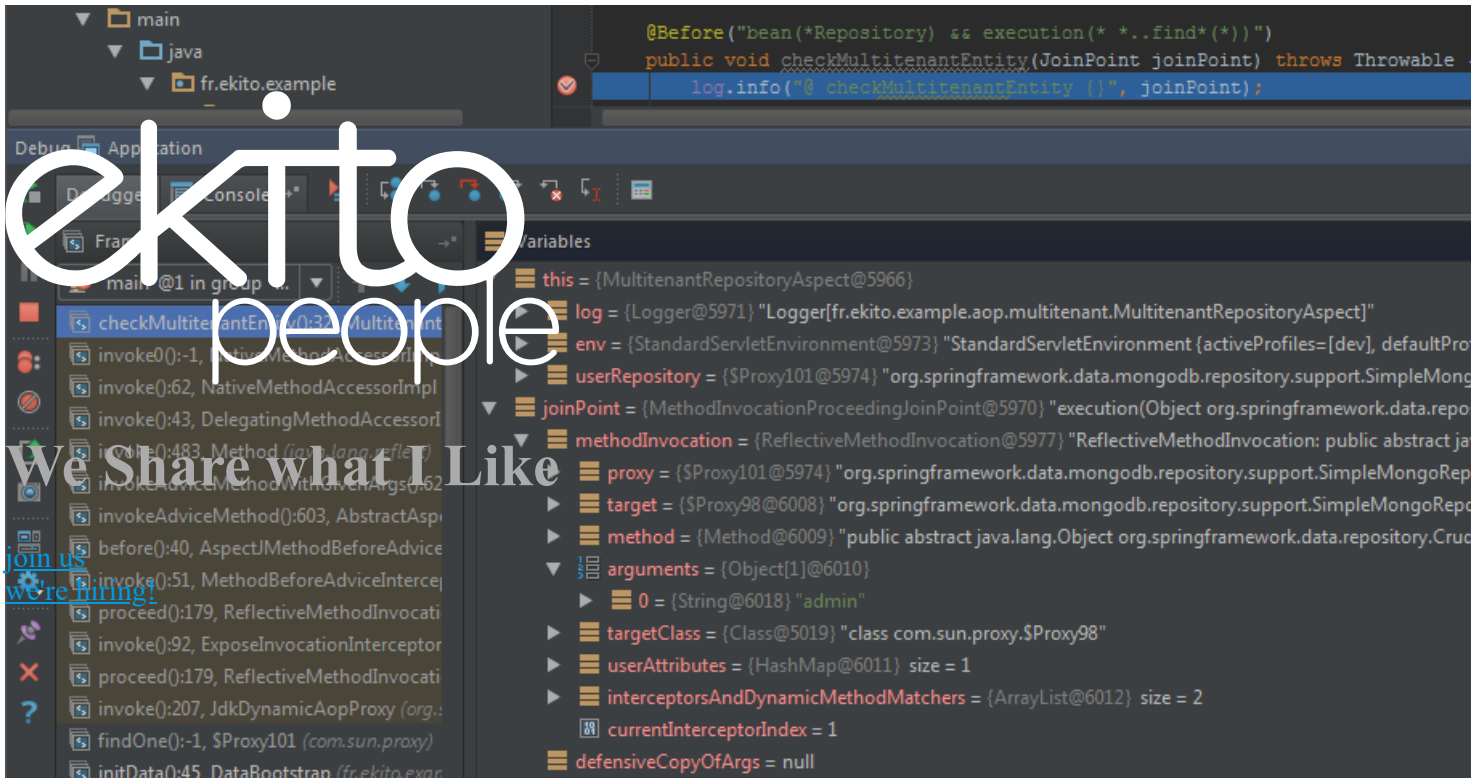
- save operations : we must ensure that the domain is injected or already set before save (avoid database constraint, domain is not-null). We can check it before any save operation and set the proper value if needed.
- find* : we must filter data across the associated domain (be sure to display data from user domain only). We could add extra criteria on "domain" field, to the current select mongodb request.
- delete : nothing … you won't delete data that you don't see.

How it goes under the hood ? We can easily capture repository beans save operations on MultitenantEntity with an aspect, and inject the right domain :

```
@Before("bean(*Repository) && execution(* *..save(*)) && args(entity)")
public void checkMultitenantEntity(MultitenantEntity entity) throws Throwable {
if (entity.getUserDomain() == null) {
    log.warn("@ checkMultitenantEntity - no group found");
    String login = SecurityUtils.getCurrentLogin();
    User user = userRepository.findOne(login);
    entity.setUserDomain(user.getUserDomain());
```

MultitenantRepositoryAspect.java

For find methods, we can intercept find method call … but how to add our domain criteria ? We can't do it from this level because it's the top level call : we have only our data and but don't have the hands on the request API. We could do it from at deeper level, but it became very complex to understand where we can cut in general. Not good 🙁



But, we can reverse the problem : we can create a new MongoTemplate bean capable of injecting criteria before run it on the system. Yeah !

MultitenantMongoTemplate.java

```
@Override
public  List find(final Query query, Class entityClass, String collectionName) {
  if (isMultitenantEntity(entityClass)) {
    injectCriteria(query);
  }
  return super.find(query, entityClass, collectionName);
}
```

```
private void injectCriteria(Query query) {
  Optional<Domain> currentDomain = SecurityUtils.getCurrentDomain();

  // check already existing present group criteria
  boolean criteriaAlreadyExists = query.getQueryObject().containsField("userDomain");

  if (currentDomain.isPresent() && !criteriaAlreadyExists) {
    Domain domain = currentDomain.get();
    query.addCriteria(where("userDomain").is(domain));
    log.info("inject domain {} in query {}", domain, query);
  //...
```

In fine, we identify the right level of data isolation and we didn't really impact our code (my collegues don't screamed) by choosing implicit domain data usage. On the github updated project (here), you can see that admin & user users don't share any data.

See our project on github : https://github.com/Ekito/spring-multitenancy

Feel free to give your feedback !

We Share what I Like

Google+
join us
we're hiring!

Published in Technology on January 5, 2015
Tags: mongoDB, spring, boot, multitenant, multitenancy, aop, spring-boot

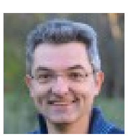
Author: Arnaud Giuliani

French Java Software Tech, create and run #java #server gears (distributed and other under the hood stuffs). Also like to make #android apps

Like it? Share it!

Tweet Like 0 Share

9 Comments

1. 

   Sylvain Wallez
   January 5, 2015 at 6:03 pm

   This approach looks nice because it avoids modifying the entire application code, but it's dangerous as it relies on a ThreadLocal at the deepest level of application code (the persistence layer) whereas the value for this ThreadLocal is set at the highest level (a servlet filter).

   All things will work nicely until you have some background tasks or start using asynchronous libraries where ThreadLocal just doesn't work and will require you to pass the SecurityContext around to reinstall it in the target thread.

   There's also a performance problem as every database operation now calls getCurrentDomain() what does a database request to get the current user domain. This can be avoided somehow by caching it in a WeakHashMap.

Also, defaulting to SYSTEM_ACCOUNT in SpringSecurityAuditorAware if no SecurityContext is found is very dangerous as it will basically give “root” rights to any background tasks where the ThreadLocal would have not been correctly initialized. It’s safer to always require it and consider it as an error if it’s not present.

So sorry, in the end it’s better to go through the necessary refactoring, and get the current domain high in the call stack and pass it around everywhere. More work, but it will avoid you to be bitten hard later by transient bugs in the in this very sensitive area that is security in a multi-tenant application.

It will also avoid things like “public static UserRepository” in SecurityUtils 🙂

Reply

2. 
Arnaud Giuliani
January 7, 2015 at 12:02 pm

Thank you very much for your appreciated feedback, Sylvain!

Please bare in mind that this post shows a refactoring approach of an existing web application. Actually, its a CRUD web application generated with the jhipster code generator. We wanted to simply add multi-tenant capabilities to our existing application in a quick and simple way.

In our case, this kind of feature injection works, because we are looking here at a CRUD application. ThreadLocal issues as you pointed out with asynchronous libraries won’t therefore be an issue in this specific context.

Yes, we reuse the Spring Security context and base some additional mechanisms on it. To avoid a bad multi-threading scenario, you can still explicitly use the domain argument (The template won’t inject the domain data if you’re providing it in your request).

Concerning the performance problem, I also thought about a caching mechanism, or even better, storing domain data directly in the security context and thus avoiding database requests to getCurrentDomain().

Effectively there is security problem. Thank you for pointing out this one. The class SpringSecurityAuditorAware is generated by jhipster and should be modified. A pull-request anyone?

My kind regards and looking forward to deepening the discussion, if you wish.

Reply

3. Pingback: Links & Reads from 2015 Week 6 | Martin's Weekly Curations

4. 
Mark Stevens
March 24, 2015 at 9:12 am

Thanks for this, makes sense. I am actually still new in this field and I’m still learning about web applications. I’ve read it from here http://www.lionleaf.com/blog/what-is-a-web-application-2/ that web apps relieve the developer from having to build separate clients for specific computers and it seems from your post that it can do even more of it. But isn’t it a bit complicated? Are there any tips that can make it simplified?

Thanks!
Mark

Reply

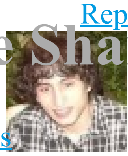
- 
  Arnaud Giuliani
  [April 3, 2015 at 8:33 am]

  Hello Mark,

  new software development are based on frameworks that allow you to build client apps with mature web patterns. Even javafx 2 extract good practices from the web (seperation of concerns : design, application & data). Take a look at [http://jhipster.github.io/] for more information about the technical stack.
  AngularJS is a very efficient technology, to help make web client apps. Take a look at it.

  Regards.

  [Reply]

5. 
[Juliano Castilho]
[March 4, 2016 at 4:21 am]

Hello Arnaud,

do you know how i can implement tenant injection in spring data querydsl?

thanks,
Juliano Castilho

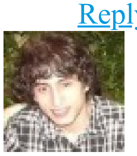
[Reply]

- 
  Arnaud Giuliani
  [March 7, 2016 at 12:14 pm]

  Hello Juliano,

  it directly depends on how you design your multitenancy. Do you do it all in the same database, as in my example below ?

  [Reply]

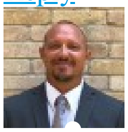
6. 
[Juliano Castilho]
[March 8, 2016 at 4:57 am]

Hello Arnaud,

I'm doing in the same database as your example. Actually, the only difference is that my repositories implements QueryDslPredicateExecutor. I'm currently injecting tenantId with AOP using ProceedingJoinPoint to modify querydsl predicate params in the service layer, only for tests.

Thanks,
Juliano Castilho

Reply

7. 

Edward Beckett
July 2[illegible], 2018 at [illegible]:28 pm

Ehe… … Strategy Pattern Alert 🙂

https://github.com/Ekito/spring-multitenancy/blob/master/updated/src/main/java/fr/ekito/example/service/DataBootstrap.java#L27-L40

Reply

We Share what I Like

▶ What do You think? Write a comment!

Author Recent Posts

join us
we're hiring!

- Android Architecture & Architecture Components – Our workshop at AndroidMakers
- Spark + Koin = ❤️
- Plus loin avec Kotlin
- Your next app will be an assistant
- Insert KOIN for Dependency Injection

Schedule a meeting with us

Arnaud Giuliani

French Java Software Tech, create and run #java #server gears (distributed and other under the hood stuffs). Also like to make #android apps

- Read Full Profile

Tweets by @arnogiu

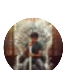
Arnaud Giuliani Retweeted

Andras Kindler
@andraskindler

Our recap of Conference for @kotliners 2018 was featured in @KotlinWeekly 🔥 This is the best way to kickstart the week, thank you! 😎

18h

Arnaud Giuliani Retweeted



Saurabh Arora
@saurabh_arora90

@saurabh_arora90

Removed dagger from the app and replaced it with @insertkoin_io Build times down by 17%. Personally, I feel the app dependencies are also a bit easier to understand now. #Android #AndroidDev #Kotlin

Aug 13, 2018

Arnaud Giuliani Retweeted

Antonio Leiva
@lime_cl

Android Developers Blog: What's new for text in Android P amp.gs/PR90 vía @google #androiddev

Embed View on Twitter

Search

Learn more about ekito