MULTI-TENANCY

in Java applications

@ladislavGazo
gazo@seges.sk

THE APPLICATION

user

|

bunch of **JavaScript**

|

service

|

domain model

IMPLICATION

user1 ... userN

|

same bunch of **JavaScript**

|

service

|

ONE database

ONE APPLICATION TO RULE THEM ALL

The term multi-tenancy in general is applied to software development to indicate an architecture in which a single running instance of an application simultaneously serves multiple clients (tenants).

This is highly common in SaaS solutions.

Isolating information (data, customizations, etc) pertaining to the various tenants is a particular **challenge** in these systems.

SPLIT ME NOW!

user1 -> BA, user2 -> BA, user 3 -> KE, user 4 -> ZA

|

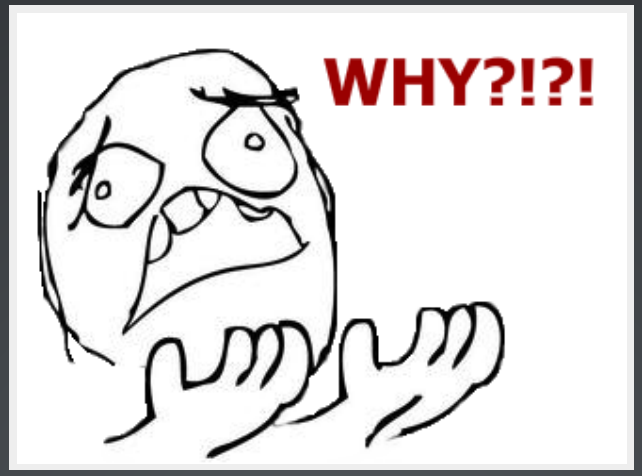
again the same **JavaScript**

|

(almost) the same **service**

|

DB (BA), DB (KE), DB (ZA)



- a lot of "tenant" dependent data - high volumes
- performance
- separation -> easier backup and **restore**
- scalability of front-end and service layer


HIBERNATE IS THE EASIEST

MULTI-TENANT CONFIGURATION

```
<property name="hibernate.multiTenancy" value="DATABASE" />
```

```
<property name="hibernate.tenant_identifier_resolver" value="sk.seges.hroddelenie.configuration.MultiTenantIdentifierResolver
```

```
<property name="hibernate.multi_tenant_connection_provider" value="org.hibernate.service.jdbc.connections.spi.
        DataSourceBasedMultiTenantConnectionProviderImpl" />
```

```
<property name="hibernate.connection.datasource" value="java:comp/env/jdbc/hr" />
```

```
<property name="hibernate.multi_tenant.datasource.identifier_for_any" value="default" />
```

SWITCH IT ON

`hibernate.multiTenancy`

CONFIGURE WHERE TO LOOK FOR THE CURRENT TENANT IDENTIFIER

```
hibernate.tenant_identifier_resolver
```

WHERE ARE MY CONNECTIONS?

```
hibernate.multi_tenant_connection_provider
```

Hibernate's DataSourceBasedMultiTenantConnectionProviderImpl utilizes JNDI lookups

POINT TO JNDI ROOT FOR DATASOURCES

```
hibernate.connection.datasource
```

... AND THE DEFAULT

```
hibernate.multi_tenant.datasource.identifier_for_any
```

So in the end = `java:comp/env/jdbc/hr/default`

BUT THERE ARE THINGS THAT DON'T WORK

- EHCache configuration from previous version is different
- Hibernate's internal schema update does not work
  - NPE !!! ..... who would have said that

MIGRATION

HIBERNATE EXPORTER

is not helpful

DO YOU HAVE ENVERS?

then you need custom exporter

IT IS EASY TO WRITE

but don't forget the AuditConfiguration

```
Ejb3Configuration jpaConfiguration = new Ejb3Configuration().configure(
        unitName, null);
Configuration hibernateConfiguration = jpaConfiguration
        .getHibernateConfiguration();

hibernateConfiguration.buildMappings();
AuditConfiguration auditConfiguration = AuditConfiguration.getFor(
        hibernateConfiguration);
```

and use EnversSchemaGenerator at last

GET RID OF NPE BUG

```
{ "hibernate.multiTenancy",
  "hibernate.tenant_identifier_resolver",
  "hibernate.multi_tenant_connection_provider",
  "hibernate.connection.datasource",
  "hibernate.multi_tenant.datasource.identifier_for_any" }
```

RUN THE EXPORTER FROM MAVEN

COMBINE IT WITH LIQUIBASE

version 3.0.6 contains feature to populate all tenant databases

```
MultiTenantSpringLiquibase
```

DON’T FORGET...

it is maintenable but generated SQL needs to be checked

existing databases must be moved to "point 0"

A BIT OF LOGGING

http://www.slf4j.org/faq.html#logging_performance

Importance of logging - using MDC to track requests

```
MDC.put("tenant", clientOrgz);
```

```
log4j.appender.stdout.layout.ConversionPattern = APP [%d{ISO8601}] - %-5p
        [%t|%X{tenant}|%X{requestId}] %F:%M %-30.30c{1} - %m%n
```

AND THE MOST BORING...

DESIGN

your app to be multi-tenant

SEND TENANT IDENTIFICATION

- in HTTP Header
  - use Filter to get it out
- in Service parameters
- using ThreadLocal variable
- via tenant-URL-based service calls
  - rmi://localhost:12345/<tenant>/<service>
- etc...

THANKS FOR LISTENING

@ladislavGazo

gazo@seges.sk