

Uma abordagem Multi Tenant baseada em Spring Boot

Consultor: Lucas Farias de Oliveira

<https://www.linkedin.com/in/luksrn/>

10/05/2017

Objetivo

O Objetivo desta consultoria é apresentar como uma integração de Multi Tenancy utilizando os frameworks, Spring Security, Hibernate 5 e Spring MVC, todos estes frameworks configurados por meio do Spring Boot 1.3.5 (Última versão estável).

O resultado da consultoria irá constar uma apresentação com tempo total de 3 horas, que deverá ocorrer no dia 10/05/2017; neste dia, serão entregues dois artefatos. O primeiro é esta documentação que descreve alguns aspectos importantes deste processo de consultoria e os principais artefatos. Já o segundo, será um código base que poderá ser utilizado pela empresa para construir o software.

O código disponibilizado poderá necessitar de alguns ajustes na integração com o sistema da empresa, tendo em vista que não houve a disponibilidade de código e base de dados para validação, porém o projeto criará componentes abstratos que poderão ser integrados de forma simples, por meio de configuração e pontos de extensão.

Descritivo do código de Exemplo

O sistema de exemplo fornecido pela consultoria tem por objetivo validar a abordagem apresentada. O escopo do sistema, acordado nas reuniões iniciais, consistem em:

- Um sistema com tela de login contendo três campos, “agência”, “usuário” e “senha”. Este seria um modelo próximo do esperado.
- Após autenticar um usuário, o mesmo será encaminhado para uma tela do sistema que irá conter as operações de Cadastrar, Remover, Atualizar e Listagem de uma entidade fictícia que irá representar um “Cliente” da agência.
- Serão apresentadas as configurações de Tenant Baseadas em Schema (Atualmente utilizada pela empresa) e Database (Apenas para avaliação de viabilidade). A abordagem por campo discriminador não será abordada, por ser mais simples, porém descrito como implementar.
- Não serão aplicados estilos CSS aos exemplos, já que a finalidade da consultoria diz respeito ao lado servidor da aplicação, sendo assim, as telas terão HTML puro.
- Não serão aplicados códigos de migração de banco de dados, pois trata-se de uma infraestrutura que a empresa já apresenta implementada via flyway, e será adaptada se necessário.

Multitenancy

O termo é, em geral, é aplicado ao desenvolvimento de software para indicar que uma instância da aplicação possui uma arquitetura que atende vários clientes (tenants). Esta é uma abordagem comum em soluções de SaaS (Software como serviço). O desafio desta abordagem é isolar as informações pertencentes a cada tenant.

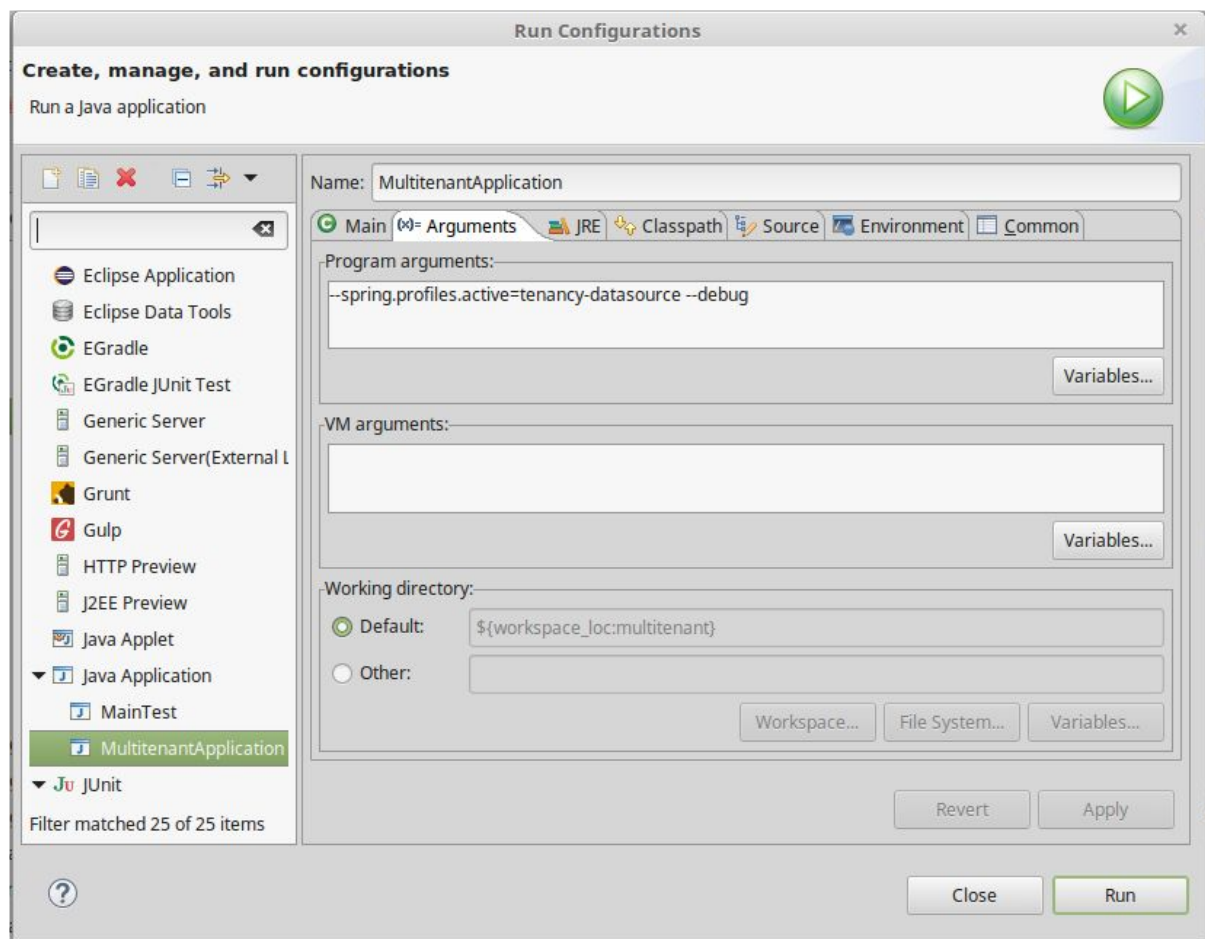
Abordagem Esquema

Cada cliente mantém os dados em esquemas diferentes de uma única instância de banco de dados. Para implementar esta abordagem, foi selecionada a lógica que; as conexões podem ser obtidas de um pool de conexão único e o schema é alterado utilizando instruções SQL como SET SCHEMA (ou similar). Esta abordagem é utilizada atualmente pela empresa, em seu sistema que está sendo migrado.

No projeto, os componentes desta abordagem são ativados pelo Spring Profile “tenancy-schema”, sendo necessário passar na inicialização o argumento

```
--spring.profiles.active=tenancy-schema
```

No Eclipse, este parâmetro pode ser passando no run configurations



ATENÇÃO: O uso de profiles foi feito no projeto de exemplo apenas para apresentar o conceito e pela viabilidade da solução, cabe a empresa manter a infra baseada em profiles ou puxar apenas os códigos relacionados a uma abordagem ou outra. Para maiores informações de profiles, visualizar

<https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-profiles.html>

Os principais componentes estão presente dentro dos pacote

com.medium.luksrn.multitenant.config.schema e as configurações gerais estão no pacote **com.medium.luksrn.multitenant**.

- **MultiTenantSchemaConfiguration** - Esta classe é a configuração da arquitetura multitenat baseada em schemas. É aqui que os componentes principais são inicializados.
- **SchemaPerTenantConnectionProvider** - Este componente é uma implementação da interface **MultiTenantConnectionProvider** do Hibernate e é responsável por recuperar uma conexão do DataSource utilizando a estratégia de tenant ID baseada em Schemas. O schema default é o "public" (E pode ser alterado se desejado).
- **MultiTenantSchemaDataSourceWrapper** - Para ser utilizado em conjunto com o JDBC Template quando o modo de Tenant for Schema based.

Abordagem Bancos de Dados

Cada cliente possui uma instância de banco de dados separada fisicamente. Nesta abordagem, nós definimos um pool de conexões por cliente e selecionamos o pool que será utilizado baseado no identificador do cliente associado ao usuário logado.

No projeto, os componentes desta abordagem são ativados pelo Spring Profile “tenancy-datasource”, sendo necessário passar na inicialização o argumento

```
--spring.profiles.active=tenancy-datasource
```

Os principais componentes estão presente dentro dos pacote **com.medium.luksrn.multitenant.config.datasource** e as configurações gerais estão no pacote **com.medium.luksrn.multitenant**.

- **DataSourcePerTenantConnectionProvider** - Este componente é uma implementação da interface **MultiTenantConnectionProvider** do Hibernate e é responsável por recuperar uma conexão do DataSource utilizando a estratégia de tenant ID baseada em Schemas. O Datasource administrativo é simbolizado pela configuração, na classe **DataSourceProperties** que informa no método *bancoAutenticacao* se ele é ou não o banco default (onde estão usuários, clientes e etc. semelhante ao schema public da abordagem de esquemas).
- **MultiTenantDatabaseDataSourceWrapper** - Quando utilizada a abordagem database, esta classe implementa um mecanismo de roteamento de banco de dados de acordo com o tenant. Ver algoritmo em **AbstractRoutingDataSource**.
- **MultiTenantDatabaseConfiguration** - configurações específicas da abordagem de bancos de dados diferentes. É aqui que os componentes principais são inicializados.

Abordagem de discriminador

Não será abordada na consultoria, porém trata-se da adição de campos extras no domínio para que o mesmo tenha a informação do tenant como coluna da tabela. Sendo assim, todas as consultas devem considerar esta informação para exibir ao usuário apenas informações da empresa dele.

É comum utilizar-se uma abordagem mista, onde a separação por bancos ou esquemas existe a nível de empresa, mas o discriminador é utilizado para particionar os dados dentro da empresa, por exemplo, usuários de unidades diferentes não devem acessar dados de outra unidade, a menos que o mesmo tenha uma permissão para tal.

Recomenda-se utilizar processadores de listeners ou JPA Callbacks para garantir que o campo seja populado sempre.

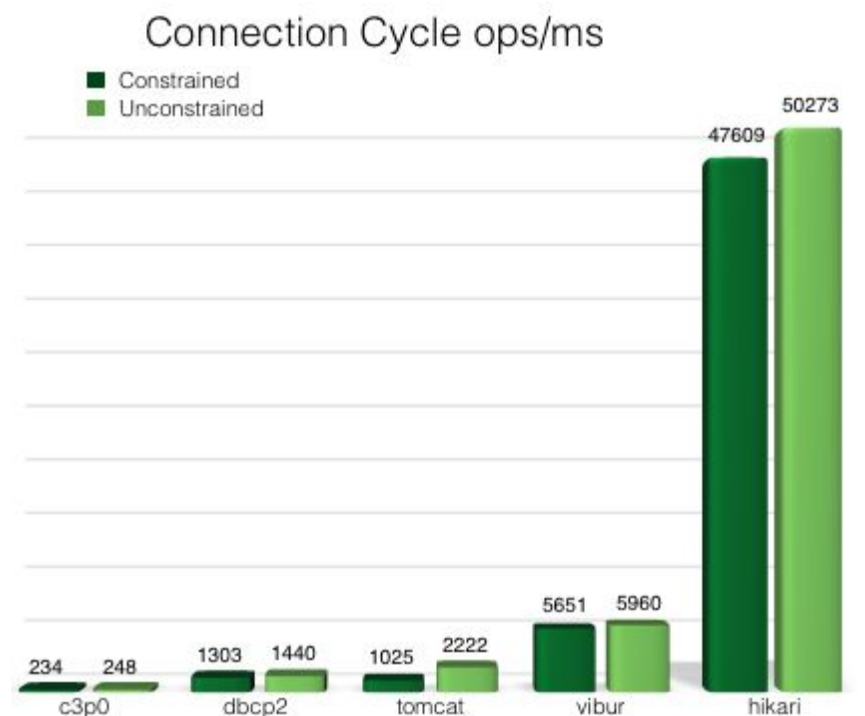
Veja

https://docs.jboss.org/hibernate/orm/current/userguide/html_single/Hibernate_User_Guide.html#events-jpa-callbacks-example

DataSource - Pool de Conexões na aplicação

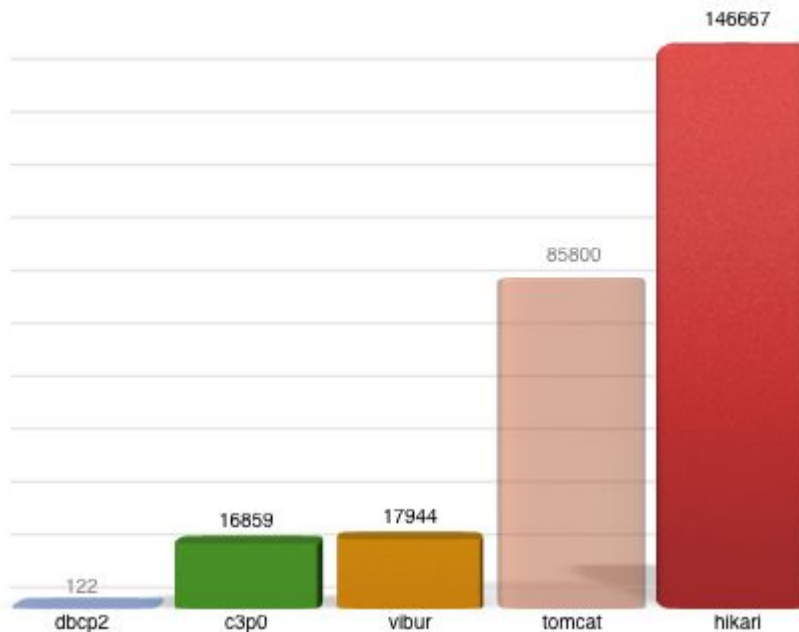
No pacote comum para as duas abordagens, temos uma interface chamada **DataSourceProvider** que é responsável por definir qual Pool de DataSources será utilizado em nosso projeto. Neste documento, recomenda-se e é exemplificado o pool de conexões HikariCP - <https://brettwooldridge.github.io/HikariCP/>

Este pool apresenta alto desempenho e testes que mostram ótimos resultados. Para exemplificar alguns dos resultados, temos o seguinte gráfico que mostra a quantidade de ciclos de abertura / fechamento de conexões que o pool conseguiu realizar. Ou seja, o número de DataSource.getConnection()/Connection.close().



Outro resultado é o número de instruções preparadas, executadas e liberadas, ou seja, Connection.prepareStatement(), Statement.execute(), Statement.close().

Statement Cycle ops/ms



Percebe-se na imagem um ótimo resultado em relação aos concorrentes mais comuns, o do tomcat e c3p0. Sendo assim, o projeto apresenta uma implementação básica sem detalhes de configuração que devem ser analisados de acordo com o interesse e situações da empresa em produção.

As configurações estão presentes em uma versão simplificada na URL

<https://github.com/brettwooldridge/HikariCP#configuration-knobs-baby> e configurações mais utilizadas no geral na url <https://github.com/brettwooldridge/HikariCP#frequently-used>.

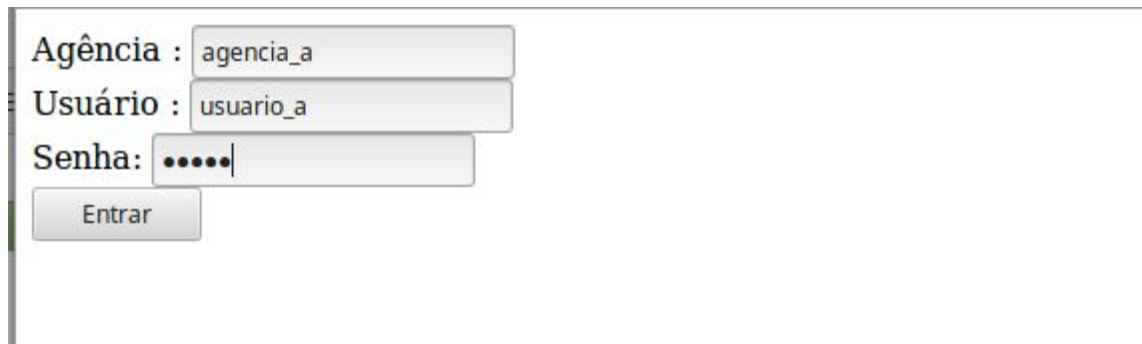
Integração com o Spring Security

Para integrar o Spring Security com o processo de vários clientes, foi utilizada a arquitetura de classes do próprio framework. Os pontos mais relevantes são:

- As classes de segurança estão no pacote **com.medium.luksrn.app.security**
- A classe de domínio que representa o usuário implementa **UserDetails**, com isso, customizando o provedor de autenticação para utilizar **AbstractUserDetailsAuthenticationProvider**
- Fornecemos um provedor de “Details” pela classe **MultitenantWebAuthenticationDetailsSource** que estende **WebAuthenticationDetails** e adiciona a informação do tenant. É por meio desta informação de “Details” que obtemos informações do usuário logado por meio da classe **SecurityContextHolder**.

Demonstrativo do sistema de exemplo

O sistema parte do principio que existe duas agências, agencia_a e agencia_b, e em cada uma existem usuários vinculados. Sendo assim, na tela de login será exibido um formulário onde o usuário irá informar a agência, usuário e senha.

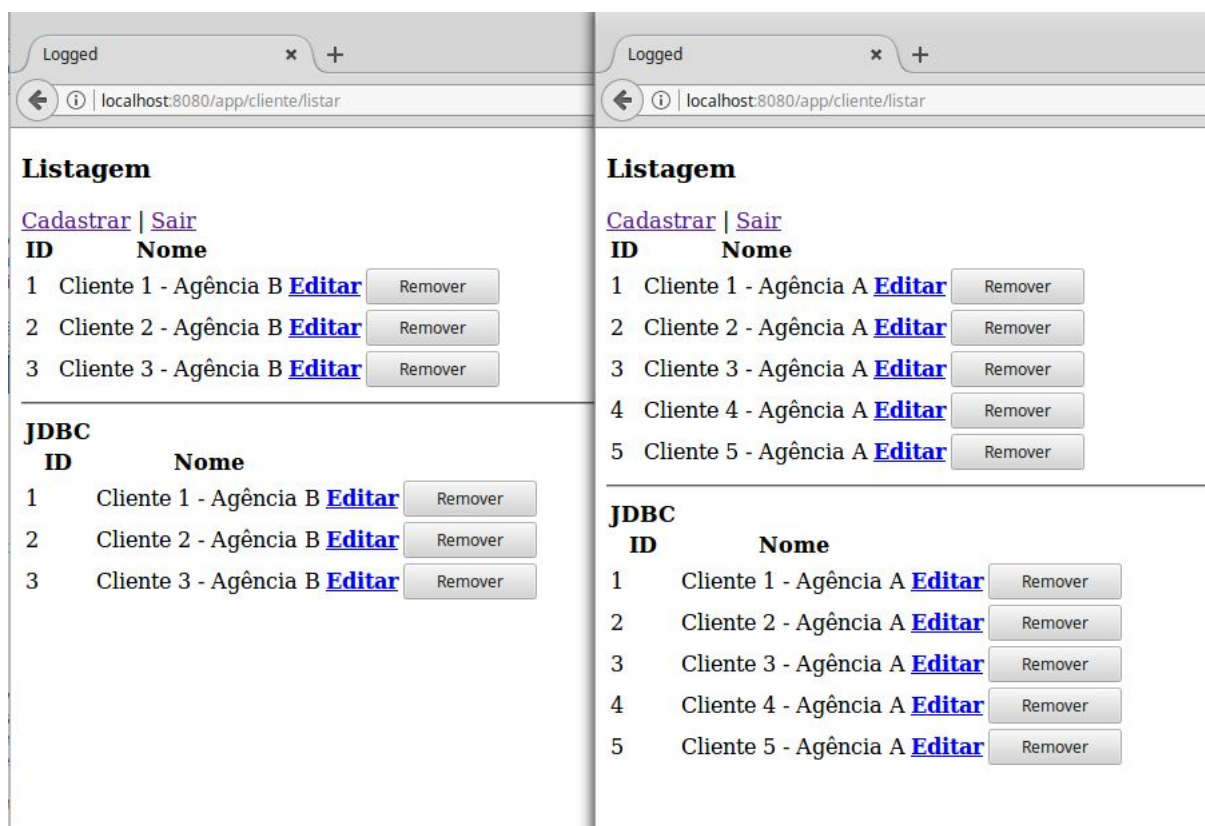


Agência :

Usuário :

Senha:

Em seguida, o usuário irá para uma tela de listagem de clientes. A imagem a seguir, apresenta uma tela lado a lado de dois usuários acessando a aplicação com roteamento em banco de dados ou esquema.



Both screenshots show the 'Listagem' page with the following structure:

[Cadastrar](#) | [Sair](#)

ID	Nome	
1	Cliente 1 - Agência B	Editar <input type="button" value="Remover"/>
2	Cliente 2 - Agência B	Editar <input type="button" value="Remover"/>
3	Cliente 3 - Agência B	Editar <input type="button" value="Remover"/>

JDBC

ID	Nome	
1	Cliente 1 - Agência B	Editar <input type="button" value="Remover"/>
2	Cliente 2 - Agência B	Editar <input type="button" value="Remover"/>
3	Cliente 3 - Agência B	Editar <input type="button" value="Remover"/>

The right browser shows data for 'Agência A' instead of 'Agência B'.

Perceba que na imagem do lado esquerdo, são exibidos apenas registros referentes a agência B (Tenant B) e do lado direito registros da empresa A (Tenant A). A primeira listagem apresenta Spring Data JPA e a segunda Spring JDBC.

Além disso, está presente uma versão simplificada da operação de **Cadastro**, **Atualização** e **Remoção**.

The image displays two browser windows side-by-side, illustrating the application's user interface for client management.

Top Window (localhost:8080/app/cliente/editar/2):

- Navigation links: [Listagem](#) | [Sair](#)
- Section title: **Form de Edição**
- Form fields: A text input labeled "Nome:" containing "Cliente 2 - Agência A".
- Buttons: "Editar" and "Reset".

Bottom Window (localhost:8080/app/cliente/cadastrar):

- Navigation links: [Listagem](#) | [Sair](#)
- Section title: **Form de Cadastro**
- Form fields: A text input labeled "Nome:".
- Buttons: "Cadastrar" and "Reset".

Pré requisitos para executar a aplicação de exemplo

- É necessário possuir uma instância de banco de dados PostgreSQL com as credenciais presentes nos arquivos **application-tenancy-datasource.yml** e **application-tenancy-schema.yml** no diretório `src/main/resources`. Estas configurações foram acordadas nas reuniões iniciais e elas devem ser obtidas de forma específica na implantação do projeto.
- Scripts para abordagem schema-based estão presentes em **V1__construcao_inicial_schema.sql**. Os demais, são da abordagem database.

Documentação Extra (JAVADOC)

O código de exemplo possui comentários em componentes essenciais e pode ser gerado pela ferramenta de build para auxiliar o entendimento da visão geral.

All Classes

Packages

[com.medium.luksrn](#)
[com.medium.luksrn.app.domain](#)
[com.medium.luksrn.app.repository](#)
[com.medium.luksrn.app.security](#)
[com.medium.luksrn.app.service](#)
[com.medium.luksrn.app.web](#)
[com.medium.luksrn.multitenant](#)

All Classes

[Agencia](#)
[AgenciaRepository](#)
[Cliente](#)
[ClienteRepository](#)
[ClientesController](#)
[CustomSuccessHandler](#)
[DataSourcePerTenantConnectionProvider](#)
[DataSourceProfile](#)
[DataSourceProperties](#)
[DataSourceProvider](#)
[GerenciarCliente](#)
[HikariCPDataSourceProvider](#)
[MultitenantApplication](#)
[MultitenantAuthenticationProvider](#)
[MultiTenantDatabaseConfiguration](#)
[MultiTenantDatabaseDataSourceWrapper](#)
[MultiTenantDatabaseProperties](#)
[MultitenantHolder](#)
[MultitenantHolderCurrentIdentifierResolver](#)
[MultitenantHolderFilter](#)
[MultitenantManager](#)
[MultiTenantSchemaConfiguration](#)
[MultiTenantSchemaDataSourceWrapper](#)
[MultiTenantSchemaProperties](#)
[MultitenantWebAuthenticationDetails](#)

OVERVIEW

PACKAGE

CLASS

TREE

DEPRECATED

INDEX

HELP

PREV

NEXT

FRAMES

NO FRAMES

multitenant 0.0.1-SNAPSHOT API

Packages

Package	Description
com.medium.luksrn	
com.medium.luksrn.app.domain	
com.medium.luksrn.app.repository	
com.medium.luksrn.app.security	
com.medium.luksrn.app.service	
com.medium.luksrn.app.web	
com.medium.luksrn.multitenant	
com.medium.luksrn.multitenant.config	
com.medium.luksrn.multitenant.config.datasource	
com.medium.luksrn.multitenant.config.schema	
com.medium.luksrn.multitenant.profiles	
com.medium.luksrn.multitenant.web	

OVERVIEW

PACKAGE

CLASS

TREE

DEPRECATED

INDEX

HELP

PREV

NEXT

FRAMES

NO FRAMES