argonavis
tecnologia e arte

JAVA EE 7
Exercícios

JAVA EE 7

Helder da Rocha

O texto foi elaborado como material de apoio para treinamentos especializados em linguagem Java e explora assuntos detalhados nas especificações e documentações oficiais sobre o tema, utilizadas como principais fontes. A autoria deste texto é de inteira responsabilidade do seu autor, que o escreveu independentemente com finalidade educativa e não tem qualquer relação com a Oracle.

O código-fonte relacionado aos tópicos abordados neste material estão em:

`github.com/helderdarocha/javaee7-course`
`github.com/helderdarocha/CursoJavaEE_Exercicios`
`github.com/helderdarocha/ExercicioMinicursoJMS`
`github.com/helderdarocha/JavaEE7SecurityExamples`

www.argonavis.com.br

R672p Rocha, Helder Lima Santos da, 1968-

Programação de aplicações Java EE usando Glassfish e WildFly.

360p. 21cm x 29.7cm. PDF.

Documento criado em 16 de outubro de 2013.

Atualizado e ampliado entre setembro e dezembro de 2016.

Volumes (independentes): *1: Introdução, 2: Servlets, 3: CDI, 4: JPA, 5: EJB, 6: SOAP, 7: REST, 8: JSF, 9: JMS, 10: Segurança, 11: Exercícios*.

1. Java (*Linguagem de programação de computadores*). 2. Java EE (*Linguagem de programação de computadores*). 3. Computação distribuída (*Ciência da Computação*). I. Título.

CDD 005.13'3

Exercícios de Java EE 7

Esta série de 12 exercícios explora as principais tecnologias, arquiteturas e padrões do Java EE 7 através de um único projeto baseado em um domínio simples. Os exercícios exploram WebServlets, JSF, JPA, CDI, JDBC, DAO, MVC, EJB, Web Services SOAP e REST em um projeto desenvolvido incrementalmente. Os exercícios são similares aos exemplos mostrados no curso, e podem ser feitos seguindo os mesmos modelos, usando o mesmo procedimento, no NetBeans ou em outro IDE como Eclipse.

Soluções completas e parciais estão disponíveis, na forma de projetos Maven (que podem ser carregadas no NetBeans ou Eclipse). É importante observar que o Maven gerencia o projeto de uma forma diferente (através do arquivo pom.xml e de menus específicos) da solução nativa do NetBeans. Se tiver dificuldade com o Maven, crie um projeto NetBeans nativo e inclua os arquivos fornecidos.

Exercícios

O código-fonte usado nestes exercícios está disponível em:

github.com/helderdarocha/CursoJavaEE_Exercicios

1 Aplicações Web (WebServlet e introdução a páginas JSF

1.1 Projeto: biblioteca-livro

Este exercício é simples, mas é importante para garantir que o seu ambiente esteja bem configurado para os próximos exercícios. Você pode importar o projeto parcial Maven no NetBeans ou começar um novo, e usar os arquivos.

Objetivo: listar livros em uma tabela HTML. Duas classes são fornecidas que representam as abstrações ilustradas acima (veja pasta /**arquivos**): **Biblioteca.java** e **Livro.java**.

Instruções: Crie um projeto Web.

a) Crie e mapeie um servlet que obtenha uma instância de Biblioteca, chame o seu método getLivros() e apresente os dados em uma página HTML.
b) Faça o mesmo usando JSF (página XHTML).

Arquivos fornecidos: **Livro.java**, **Biblioteca.java** e **index.xhtml** (este último pode ser usado como ponto de entrada da aplicação Web. Ele já está previamente configurado com dois links (mapeamentos sugeridos para o servlet e página JSF).

Arquivos extras, parcialmente implementados estão na pasta /**extras**: **biblioteca-page.xhtml** (coloque na pasta webapp do projeto) e **BibliotecaServlet.java** (coloque no pacote correspondente na pasta de código-fonte).

Um projeto Maven, configurado para uso de servlets e JSF, contendo esses arquivos está disponível em /**projeto-parcial**. É preciso editar biblioteca-page.xhtml e BibliotecaServlet.java para concluir o projeto.

A solução (projeto Maven) está na pasta /**solucao**.

Dicas:

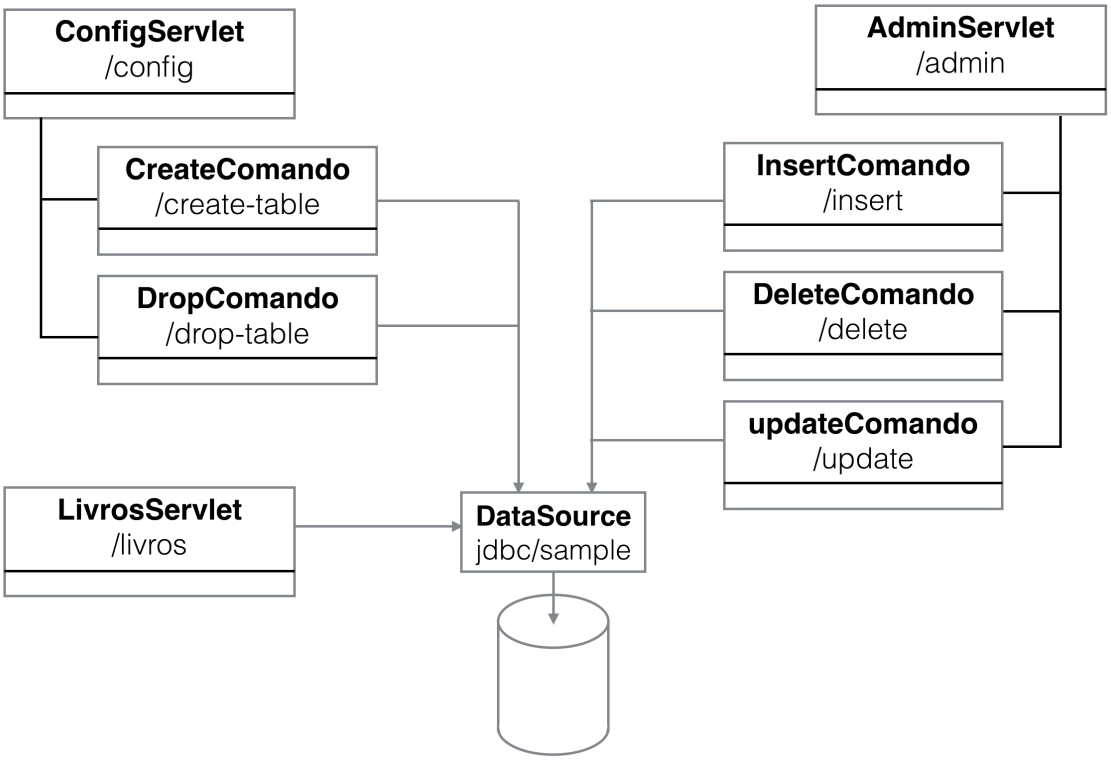
1) Use @Inject para disponibilizar Biblioteca no servlet. Como ele está registrado como um bean CDI (@RequestScoped) ele pode ser inserido dessa forma.
2) Biblioteca está registrada para uso via JSTL (@Named). Use <h:dataTable> ou <ui:repeat> e chame propriedades do bean bean via JSTL #{bean.propriedade}

2 Acesso a banco de dados via Web

2.1 Projeto: biblioteca-livro-jdbc

Objetivo: manter uma lista de livros em um banco de dados, e usar componentes Web para inserir, atualizar, remover e listar os livros. Como o objetivo deste exercício é JDBC, a maior parte da aplicação está implementada. Crie ou adapte um projeto Web e use as classes fornecidas em **/arquivos**, e **/extras** ou inicie com o projeto Maven em **/projeto-parcial**.

O diagrama abaixo ilustra os componentes da aplicação.



A aplicação Web utiliza o padrão de design *Front Controller*, onde um servlet despacha requisições para comandos e recebe como resposta informações de navegação. Cada comando possui um método execute() que recebe objetos HttpServletRequest e HttpServletResponse, retornando uma string, que corresponde à URL da próxima View a ser mostrada. Veja o código de cada servlet (que está pronto) e da interface Comando.

Analise o código existente antes de iniciar. Apenas três arquivos precisam ser editados ou criados: **InsertComando.java** (SQL insert), **UpdateComando.java** (SQL update) e **LivrosServlet.java** (SQL select). Os outros três comandos (que implementam o SQL delete, create table e drop table) estão prontos.

O primeiro ConfigServlet (mapeado como "**/config/***") tem a finalidade de configurar o ambiente, criar a tabela ou destruí-la. Os comandos devem ser passados como informação "extra path", ou seja, como parte da URL, por exemplo:

*http://localhost:8080/contexto/config/**create-table*** para criar a tabela e inserir cinco livros iniciais, e *http://localhost:8080/contexto/config/**drop-table*** para derrubar a tabela.

O segundo servlet AdminServlet ("**/admin/***") é similar ao primeiro e possui três comandos: um para inserir um novo livro ("**/insert**"), outro para remover um livro ("**/delete**"), outro para atualizar um livro existente ("**/update**"). Todos esses comandos requerem parâmetros, que devem ser passados no query-string (ou campos de formulário HTTP). Os parâmetros são lidos nos servlets.

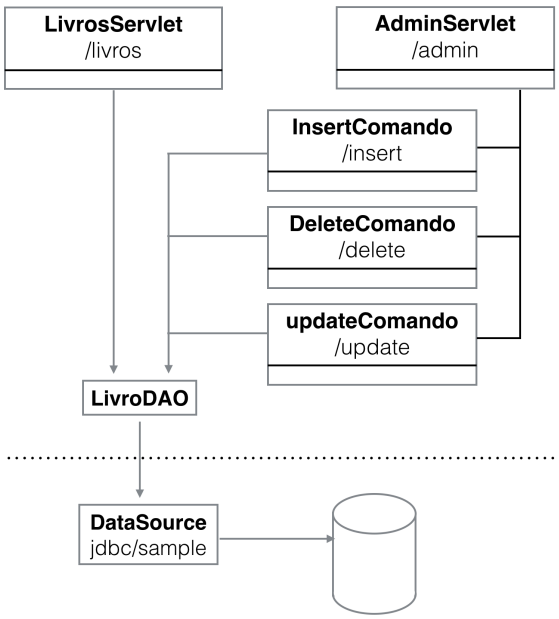
O terceiro servlet LivrosServlet ("**/livros**") lista todos os livros em uma tabela, e contém botões para editar e remover.

Como no exercício anterior, há uma solução (pasta **/solucao**), projeto parcialmente criado (**/projeto-parcial**), e, para quem quiser montar um projeto do zero, arquivos prontos em **/arquivos** e arquivos que precisam ser editados em **/extras**.

3 Aplicação Web com banco de dados e DAO

3.1 biblioteca-livro-dao

Objetivo: refatorar o exercício anterior para *isolar* a camada JDBC de forma que a camada Web (servlets e comandos) não dependam da API JDBC (veja ilustração ao lado).



Os servlets e comandos deverão usar a interface abaixo para ter acesso aos dados:

```
public interface LivroDAO {
    Livro findByID(int id) throws IOException;
    Livro findByISBN(String isbn) throws IOException;
```

```
    Collection<Livro> getLivros() throws IOException;
    int insert(Livro livro) throws IOException;
    void update(Livro livro) throws IOException;
    void delete(Livro livro) throws IOException;
}
```

O objetivo deste exercício é implementar o DAO (copiar e adaptar o código JDBC usado nos comandos, e incluí-los em métodos do DAO), e depois adaptar os comandos e servlets de forma que eles usem o DAO em vez do DataSource.

Este exercício pode ser feito a partir do exercício anterior. Altere os construtores dos comandos para que passem um **LivroDAO** (em vez de **DataSource**), adapte **AdminServlet** para injetar o DAO e passar para cada comando na inicialização (não é preciso incluir código CDI nas classes e interfaces se um arquivo **beans.xml** com a opção *bean-discovery-mode="all"* estiver presente em WEB-INF). Implemente a interface do DAO (aproveite o código JDBC dentro dos comandos) e altere os comandos para chamar os métodos do DAO.

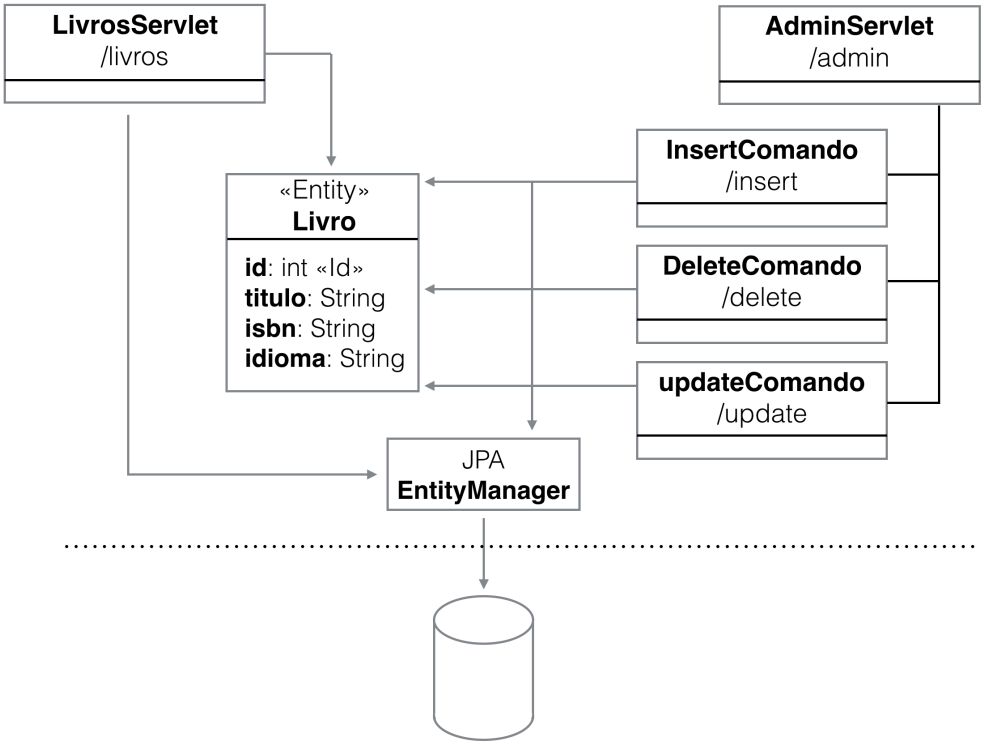
Os arquivos que não precisam ser alterados estão na pasta **/arquivos**. Arquivos alterados estão na pasta **/extras**. Um projeto Maven parcial está em **/projeto-parcial** e a solução na pasta **/solucao**.

Antes de começar analise e entenda o código que está pronto (HTML, XHTML, Java, arquivos XML).

4 Acesso a banco de dados via Web

4.1 biblioteca-livro-jpa

Objetivo: alterar o exercício anterior e substituir a camada de persistência DAO por JPA. Para isto o objeto Livro deve ser transformado em uma entidade (através de anotações @Entity e @Id), e o DAO deve ser implementado com chamadas a métodos do EntityManager.



A interface não precisa mais declarar as exceções usadas no DAO, portanto pode ser implementada de forma mais simples (o nome anterior pode ser mantido):

```
@RequestScoped
public interface LivroService {
    Livro findByID(int id);
    Livro findByISBN(String isbn);
    Collection<Livro> getLivros();
    int insert(Livro livro);
    void update(Livro livro);
    void delete(Livro livro);
}
```

A anotação @RequestScoped (do pacote javax.enterprise.context) permitirá que o serviço seja injetado nos servlets e usado nos comandos.

É preciso incluir um arquivo **persistence.xml** corretamente configurado para o DataSource em "jdbc/sample" como "JTA". A entidade Livro deve ser declarada. Este arquivo pode ser gerado pelo IDE e está disponível em /arquivos.

Na implementação de LivroService é necessário injetar um EntityManager (use @PersistenceContext) com o unitName correspondente ao nome da unidade de persistência em persistence.xml. Os métodos de EntityManager devem ser chamados em um contexto transacional. Isto pode ser feito de duas maneiras: 1) Injetando um @Resource UserTransaction ut, e delimitando as chamadas com ut.begin() e

ut.commit(), ou anotando cada método com o aspecto @Transactional do CDI, que faz a mesma coisa.

Os arquivos que precisam ser alterados neste exercício são a implementação de LivroService (LivroDAO), que precisa ser criada, o arquivo Livro, que precisa ser transformado em Entity, e persistence.xml que precisa ser criado. Se o nome de LivroDAO não for alterado, não é necessário alterar mais nada.

A solução está na pasta **/solucao**. Em **/projeto-parcial** há um projeto Maven onde falta apenas alterar os arquivos **Livro.java** e **LivroJPAService.java**. Esses arquivos estão na pasta **/extras**. Os outros arquivos, que não precisam ser alterados, estão na pasta **/arquivos**.

5 Aplicação Web com EJB, CDI e JPA

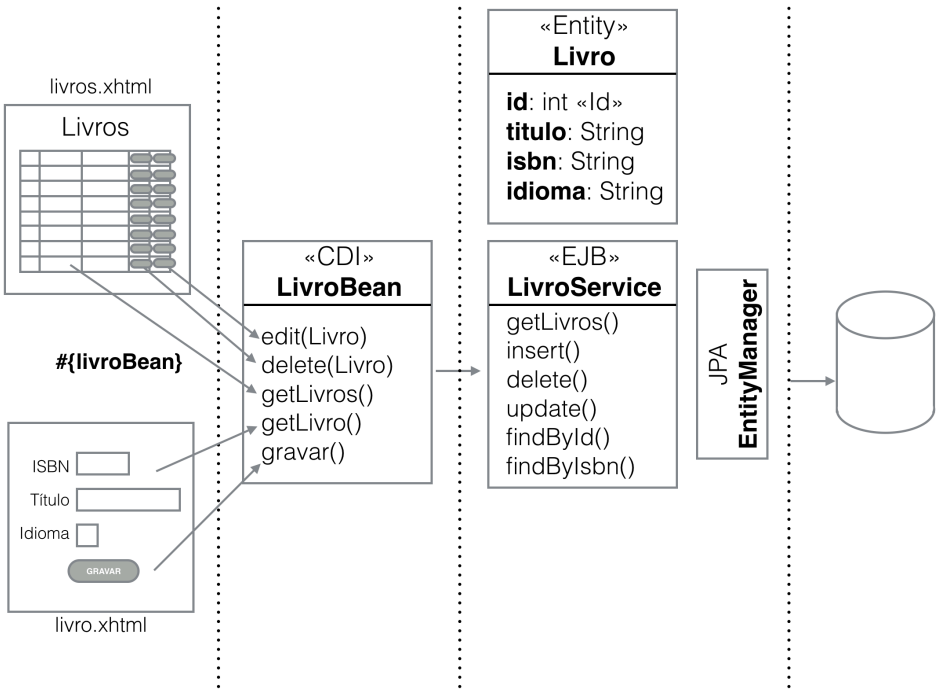
5.1 biblioteca-livro-ejb

Objetivos: Separar as responsabilidades da camada Web, dados e negócios, substituindo os comandos por backing beans (CDI) mapeados a propriedades e ações de formulários JSF, e isolando a camada de dados atrás de uma fachada de serviços EJB.

Neste exercício, o código dos comandos irá migrar para métodos de um LivroBean (bean CDI), que também irá conter uma propriedade representando uma entidade cujos campos serão mapeados a um formulário, para inserção e edição de livros. A responsabilidade do LivroBean será interligar a camada Web com a camada de negócios. Ele não irá realizar operações de persistência.

A camada de negócios será implementada através de um EJB Stateless Session Bean, via interface @Local, que oferece uma fachada de serviços que será injetada (via @EJB) no LivroBean. O EJB usará o EntityManager para implementar os métodos da camada de persistência. Como os métodos do EJB serão transacionais por default, seu código ficará muito mais simples com apenas três classes: Livro (Entity), LivroJPAService (EJB) e LivroBean (CDI), mais a interface LivroService (EJB).

O diagrama abaixo ilustra a arquitetura da aplicação.



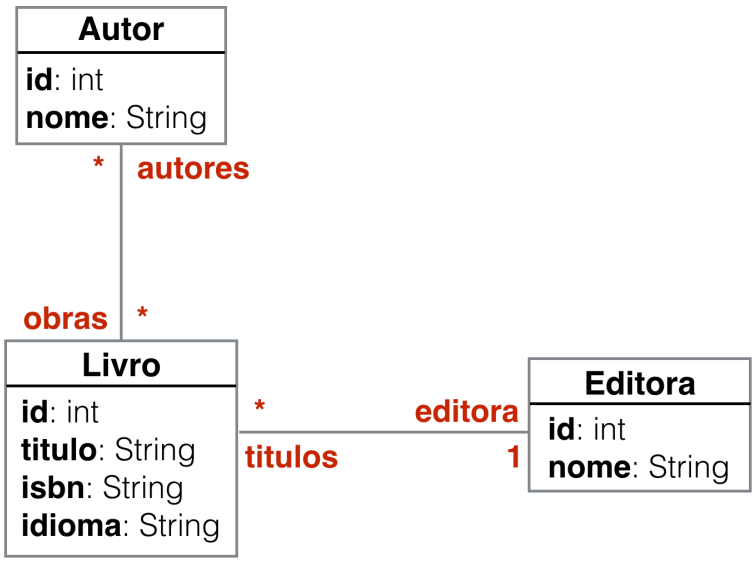
Construa a aplicação a partir do exercício anterior, ou a partir de um novo projeto. Opcionalmente aproveite os arquivos HTML e as classes prontas (pasta **/arquivos**) ou semi-prontas (pasta **/extras**), ou ainda o projeto (Maven) parcialmente concluído em **/projeto-parcial**. A solução está disponível na pasta **/solucao**.

6 Relacionamentos com JPA

6.1 Projeto: biblioteca-jpa

Objetivos: 1) Enriquecer o modelo de domínio da aplicação com duas novas classes: Autor e Editora, e alterar a classe Livro para incluir uma lista de autores, e uma editora:

2) Configurar Autor e Editora como entidades (use anotações e configure o persistence.xml se necessário), e implementar relacionamentos JPA entre eles.



3) (opcional) Fornecer uma (ou mais) fachada de serviços EJB (ex: uma interface Biblioteca, ou interfaces LivroService, AutorService, etc. para cada entidade) com uma série de serviços que serão disponibilizados pela camada de negócios. Incluir listas de livros, autores e editoras, pesquisa por ISBN para livros e operações CRUD. Este exercício está resolvido no /projeto-parcial.

4) (opcional) Implementar uma interface simples usando JSF e backing beans que permitam a) inserir novos livros, autores e editoras, b) listar livros, autores e editoras existentes, c) selecionar autores e editoras existentes na inserção e edição de livros. Não precisa ser uma interface interativa (pode ser um servlet para criar os livros, autores e editoras, e uma página XTHML para listar os resultados, como no exemplo AppWebCorrida). Este exercício está resolvido no /projeto-parcial com uma solução interativa.

O foco deste exercício é explorar relacionamentos entre entidades, portanto, os itens 3 e 4 são opcionais (se não quiser criar EJBs e as interfaces JSF e backing beans aproveite os arquivos disponíveis em **/arquivos** e **/extras**.)

Mantenha o exercício simples. Crie os relacionamentos e escreva uma pequena aplicação de testes para verificar que os relacionamentos estão funcionando corretamente. Não é preciso configurar cascade nos mapeamentos (mas verifique que os objetos estejam persistentes antes de criar as associações).

Não há uma única maneira de realizar este exercício. O mais importante é utilizar uma arquitetura que separe eficientemente as camadas de dados, de negócios e Web.

A solução proposta tem como objetivo principal ser didática. Fique à vontade para melhorá-la se desejar.

Construa a aplicação a partir do exercício anterior, ou a partir de um novo projeto. Opcionalmente aproveite os arquivos HTML e as classes prontas (pasta **/arquivos**) ou semi-prontas (pasta **/extras**), ou ainda o projeto (Maven) parcialmente concluído em **/projeto-parcial**. A solução está disponível na pasta **/solucao**.

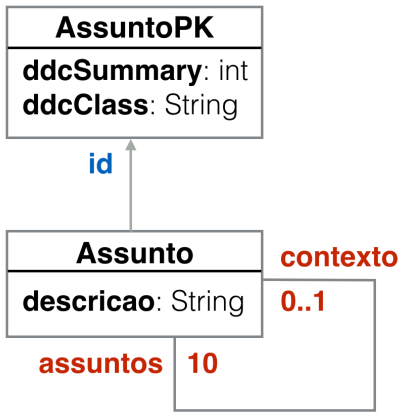
7 Exemplo usando Singleton EJB, Relacionamentos JPA, formulários JSF com Ajax e escopo de sessão CDI

7.1 biblioteca-ejb

Objetivos do exercício: 1) Configurar e instalar um exemplo parcialmente implementado que usa um Singleton para configurar uma base de dados a partir de um arquivo XML. 2) Incluir um novo relacionamento JPA na aplicação. 3) Configurar as interfaces JSF da aplicação para listar e incluir os relacionamentos nos objetos existentes.

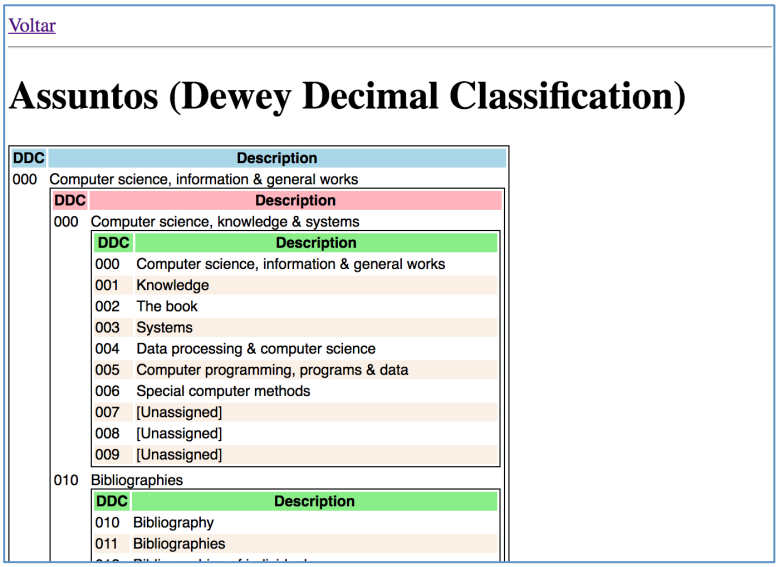
Este exemplo explora os seguintes tópicos de Java EE: EJB (Singleton Session Bean), JPA (chaves compostas, cascade, relacionamentos), JSF (ajax, conversores, validação) e CDI (@ConversationScoped), além do uso da API JAXP para ler XML.

Este exercício é mais complexo que os anteriores, mas está parcialmente pronto. Invista algum tempo para instalar e executar a versão disponibilizada em **/projeto-inicial**. A diferença entre ele e o exemplo dos exercícios anteriores é a presença de mais uma entidade: Assunto, que tem um relacionamento um-para-muitos consigo mesmo, e usa uma chave composta que não é gerada automaticamente:

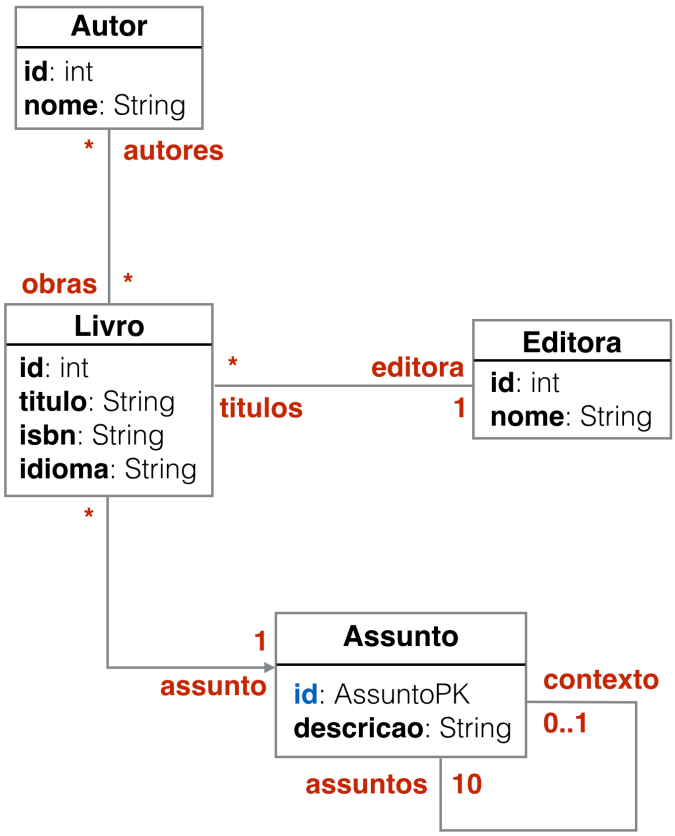


O exemplo fornecido contém um componente de configuração que carrega um XML contendo 1110 assuntos (classificação Dewey) e armazena no banco de dados. A carga usa a API JAXP e leitura baseada em eventos (SAX) para carregar a árvore de assuntos em um mapa em cuja raiz estão os dez assuntos básicos (000 a 900) do primero nível (chamado de *summary*). Cada assunto possui dez sub-assuntos (*summary 2*), que têm os assuntos básicos como contexto. E cada sub-assuntos tem mais dez sub-assuntos (*summary 3*). O XML é processado sequencialmente no método configure() do AssuntoEJB, que usa um handler (AssuntosSaxHandler) para montar o mapa. Ao final, cada assunto raiz é persistido usando em.persist() e os seus sub-assuntos são persistidos transitivamente via cascade.

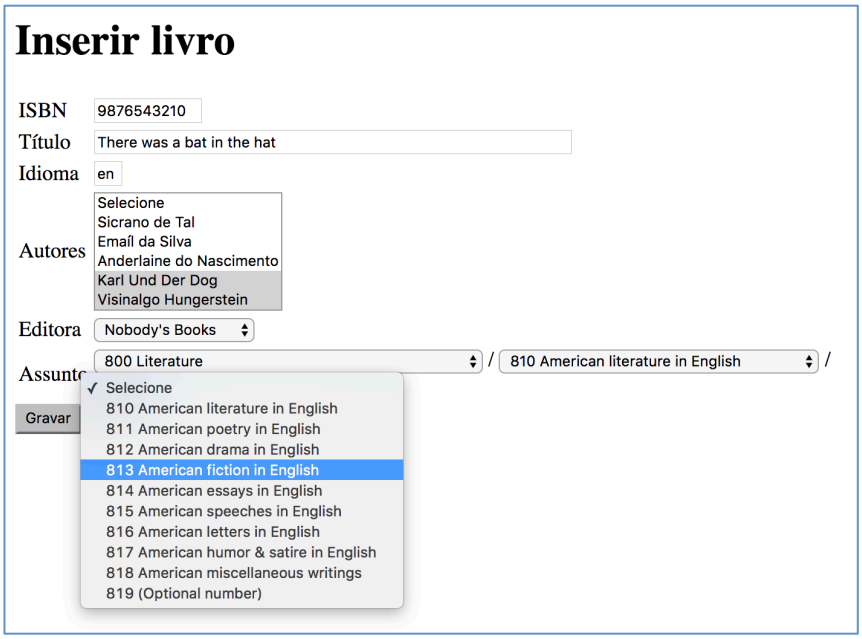
Após a configuração (que é feita uma única vez), os assuntos podem ser listados em uma página. Um dataTable (JSF) é usado para listar os assuntos depois que a aplicação é configurada.



O exercício consiste em alterar a aplicação de forma a introduzir um novo relacionamento *Livro-Assunto*, como ilustrado no diagrama abaixo:



Altere a entidade **Livro** e inclua um relacionamento **ManyToOne** com a entidade **Assunto**. Quando terminar, configure os views (páginas XHTML e LivroBean) para que um assunto possa ser adicionado a um livro (descomente os trechos de código indicados em LivroBean.java, livros.xhtml e livro.xhtml).



Livros

ID	ISBN	Título	Idioma	Editora	Autores	Assunto		
19752	1234567890	Era uma vez um Bicho	pt	Livros Hipopótamo	Anderlaine do Nascimento	592 Science / Animals (Zoology) / Invertebrates	Remover	Editar
19801	3344556677	The Big Foot	en	Nobody's Books	Karl Und Der Dog	813 Literature / American literature in English / American fiction in English	Remover	Editar
19851	9786656656654	Bugs and Bats	en	Nobody's Books	Karl Und Der Dog	590 Science / Animals (Zoology) / Animals (Zoology)	Remover	Editar
19901	4343434343	O Espaguete Voador	pt	Livros Hipopótamo	Visinalgo Hungerstein	299 Religion / Other religions / Religions not provided for elsewhere	Remover	Editar
19952	9786663636363	Elfish para iniciantes	pt	Livros Hipopótamo	Sicrano de Tal	491 Language / Other languages / East Indo-European & Celtic languages	Remover	Editar
19953	9876543210	There was a bat in the hat	en	Nobody's Books	Karl Und Der Dog , Visinalgo Hungerstein	827 Literature / English & Old English literatures / English humor & satire	Remover	Editar

Inserir

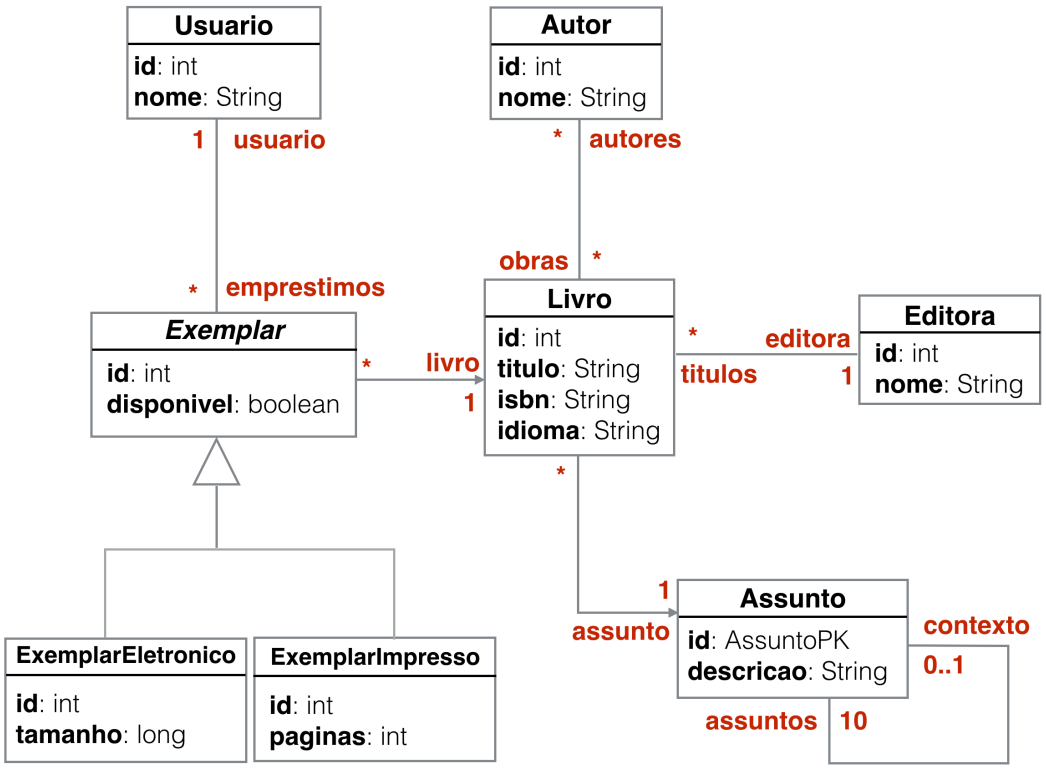
A página livro.xhml realiza requisições assíncronas (Ajax) para atualizar dinamicamente os menus de sub-assuntos, quando o contexto é selecionado. O estado dos dois primeiros menus é armazenado em propriedades (nível1 e nivel2) no LivroBean, e o último atualiza a entidade. O tag <f:ajax> tem um atributo render que informa quais os componentes que precisam ser atualizados após a seleção. Para que o estado do LivroBean seja mantido através de várias requisições, é preciso que esteja em um escopo de sessão, configurada via @ConversationScope (que inicia quando o bean é carregado, e termina quando o formulário é enviado ou cancelado). A entidade Livro também está usando anotações de validação (API Bean Validation) cujas mensagens de erro são mostradas através dos tags <h:message> da página livro.xhtml.

Uma solução está disponível na pasta **/solucao**.

8 Mapeamento de Herança em JPA

8.1 Projeto: biblioteca-heranca-jpa

Objetivos: Criar uma camada nova na aplicação para possibilitar o controle de exemplares e empréstimo de livros. A camada inclui as novas entidades **Exemplar** (abstrata) e **Usuario**,. Há dois tipos concretos de Exemplar: *ExemplarImpresso* e *ExemplarEletronico*. É necessário realizar o mapeamento da herança em JPA. O diagrama das entidades da aplicação está ilustrado abaixo.



O exercício consiste em transformar em entidades e configurar o mapeamento das classes **Usuario**, **Exemplar**, **ExemplarEletronico** e **ExemplarImpresso**. As classes (beans, EJBs, conversores) e páginas Web que possibilitam a execução e teste da aplicação estão fornecidas em **/projeto-inicial**. Analise o código, construa e execute o projeto antes de iniciar. Quando terminar, descomente as linhas indicadas em ExemplarBean e tente listar e criar Exemplares e Usuarios.

Ao concluir o exercício será possível criar um ou mais exemplares para livros existentes, e listar os exemplares disponíveis, informando número de páginas ou tamanho, dependendo do tipo. Escolha qualquer uma das estratégias de herança disponíveis para JPA (se desejar, experimente com todas). Nos próximos exercícios realizaremos consultas e empréstimo de livros usando essa aplicação.

9 Queries em JPA com JPQL e Criteria

9.1 Projeto: biblioteca-query

Objetivos: Partindo da aplicação desenvolvida no exercício anterior, criar métodos para realizar diversos tipos de consultas em exemplares, livros, autores, editoras, usuários e assuntos, usando as API Criteria e a linguagem JPQL.

Este exercício possui duas partes. A primeira consiste em criar consultas simples e estáticas, que serão exibidas em uma página. Na segunda parte as pesquisas devem ser incorporadas na aplicação para filtrar listas de livros, autores, editoras, etc.

a) A primeira parte (**/projeto-parte-1**) envolve dados estáticos. Veja na página queries-9.xhml e QueryBean.java as 6 consultas JPQL que devem ser implementadas. Implemente uma de cada vez e veja os resultados. O único arquivo que precisa ser alterado é **QueryBean.java**.

Para testar os resultados é preciso ter dados de teste. Se desejar use a infraestrutura existente para inserir livros, autores, editoras, ou use os links disponibilizados para configurar um ambiente de testes na página setup-testes.xhtml. Um bean (ConfigBean) e um EJB (TestServiceEJB) foram disponibilizados para esvaziar as tabelas e incluir dados de teste. Esvazie as tabelas e depois aperte o segundo botão da página setup-testes.xhtml para criar os dados de teste.

Observação: como não foi implementado mapeamento de cascade entre Livro, Autor, Editora e Exemplar, a remoção das entidades em setup-testes.xhtml precisa ser feita em ordem: primeiro Autor, depois Livro, Exemplar, Editora. Fica como **exercício** implementar o cascade-delete para evitar esse problema (aproveite e implemente também o cascade-persist e merge, e depois simplifique o código no TestServiceEJB.java onde indicado.)

b) Na segunda parte (**/projeto-parte-2**) devem ser implementadas pesquisas interativas.

1) Incluir um campo de texto na lista de autores para filtrar pelo nome.
2) Incluir um campo de texto na lista de editoras para filtrar pelo nome.
3) Incluir campos de texto na lista de assuntos para filtrar pela descrição e pelo código, além de um menu para selecionar qual índice usar. (resolvido)
4) Incluir campos de texto na lista de livros para filtrar pelo título, autor, editora e assunto, e um menu para filtrar pelo idioma. (resolvido)

A pesquisa deve alterar a lista de itens exibidos em tempo real (com delay de um segundo, para campos de texto, portanto é preciso que os componentes usem Ajax, renderizem novamente a tabela para cada alteração, e que os beans preservem o estado dos dados durante as várias requisições que durar a pesquisa (use @ConversationScope)

Nesta versão estão resolvidos os exercícios 3 e 4 acima, que usam a API Criteria. Não deixe de analisar o código das classes e páginas envolvidas para entender o seu

funcionamento, antes de tentar fazer os outros. As classes incluem EJBs (onde são implementados os queries JPQL ou Criteria – veja apostila de JPA) e managed beans (onde as consultas são delegadas para EJBs, e dados são mapeados a formulários e componentes JSF). As páginas são acopladas aos managed beans onde os componentes são mapeados. Veja mais informações sobre os componentes usados e o uso de <f:ajax> na apostila de JSF.

Para os exercícios desta seção (1 e 2) os arquivos a serem alterados são AutorBean, EditoraBean, AutorEJB, EditoraEJB, autores.xhtml, editoras.xhtml. Procure as indicações (comentários) em cada arquivo para mais detalhes, e use os exercícios resolvidos como modelo.

A solução destes exercícios está em **/solucao**. Se você achar que o exercício está muito simples e quiser um desafio maior, use como ponto de partida a solução do exercício anterior.

10 Stateful Session Beans

10.1 Projeto: biblioteca-session-jpa

Objetivos: Implementar a funcionalidade de empréstimo de livros, autenticação e autorização. 1) Escrever queries para pesquisas envolvendo exemplares e livros. 2) Criar uma cesta de empréstimos, onde um usuário pode selecionar livros que deseja tomar emprestado. Os livros ficam temporariamente indisponíveis até que ele cancele ou confirme o empréstimo. Livros emprestados ou indisponíveis não aparecem para seleção. 3) Criar uma lista de livros emprestados por usuário, onde ele pode devolver livros selecionados. 4) Criar um mecanismo de login (falso, sem autenticação real) para capturar o nome do usuário logado em uma sessão.

Este exercício inclui mudanças em várias classes, em relação ao exercício anterior, portanto, como ponto de partida está fornecido um parcialmente resolvido. Abra e analise todos os documentos XHTML e classes antes de começar, para entende-las.

Nesta aplicação há duas abstrações envolvendo livros: Livro e Exemplar. Livro é identificado por um ISBN (no mundo real, o ISBN é usado para identificar a Edição de um livro, mas simplificamos isto na abstração). Exemplar representa um objeto que pode ser emprestado. Livro representa a informação e Exemplar a forma como ela é distribuída (tipo eBook ou impresso, número de páginas, tamanho em bytes). Cada livro criado tem pelo menos um exemplar (criado junto com o livro), mas é possível

criar exemplares adicionais posteriormente. A página de testes inicialmente cria vários livros e exemplares.

O Exemplar possui uma propriedade disponível, que indica que o livro pode ser listado. Sempre que um usuário seleciona um livro e põe em sua cesta de empréstimos, ele se torna indisponível. Se o usuário desistir, a cesta é esvaziada, e o livro torna-se novamente disponível. As páginas que listam exemplares para empréstimo mostram apenas os livros disponíveis.

Quando o usuário estiver satisfeito com sua cesta de empréstimos, ele pode confirma-lo. A confirmação requer a criação de uma associação entre o usuário e o livro. Esta associação é realizada de forma persistente e tem tempo indeterminado. Outra página da aplicação permite que o usuário veja os livros que tomou emprestado. Ele pode selecionar, nesta página, livros que pretende devolver (que já ficam disponíveis, mas não aparecem na lista enquanto o usuário não confirmar a devolução, desvinculando-se do Exemplar.

O Usuário possui um nome e uma senha. Ao criar um usuário uma senha ("java") é atribuída a ele automaticamente. O acesso à aplicação é interceptado por um filtro (WebFilter) que verifica se há um usuário na sessão. Se não houver, ele redireciona à página de login onde o usuário digita nome e senha. Se acertar nome e senha, ele é redirecionado ao menu principal, onde terá acesso às operações da aplicação, ou à possibilidade de fazer logout. Este login não usa mecanismos de autenticação do Java EE (que dependem de recursos proprietários do servidor de aplicação). Isto será explorado em outro exercício.

Descrição dos exercícios:

a) Login. Arquivos em **projeto-parcial-1/**. Implementar uma página/bean/serviço de login que registre um usuário na sessão, e um filtro para redirecionar para a página se o bean estiver na sessão. **Este exercício está resolvido.**
b) Queries. Arquivos em **projeto-parcial-2/**. Implementar queries estáticos de acordo com as instruções no arquivo **QueriesBean.java**. Alguns queries só terão resultados depois de implementados os exercícios (c) e (d). Implemente os queries em *QueryBean.java* e veja os resultados acessando *queries-10.xhtml*.
c) Cesta de empréstimos. Arquivos em **projeto-parcial-3/**. Implementar código para que usuário possa selecionar exemplares e transferi-los para um Map, mantido por um EJB. Exemplares que estiverem no Mapa devem ser marcados como indisponíveis (esse estado deve ser sincronizado, para que outros

usuários vejam). Se a cesta for esvaziada, todos os exemplares devem ser marcados como disponíveis. Os exercícios estão indicados com comentários nos arquivos: *emprestimo.xhtml, cesta-emprestimo.xhtml, CestaLivrosEJB.java, ExemplarEJB.java, EmprestimoBean.java*.

d) Empréstimo e devolução. Arquivos em **projeto-parcial-4/**. Implementar código para realizar o empréstimo (persistir objetos da cesta, vincular a usuário e esvaziar a cesta) e devolução (desvincular usuário e marcar exemplar como disponível). Os exercícios estão indicados com comentários nos arquivos: *emprestados.xhtml, cesta-emprestimo.xhtml, CestaLivrosEJB.java, ExemplarEJB.java, EmprestimoBean.java*.

11 Web Service SOAP

11.1 biblioteca-ws-soap, assunto-soap-client, assunto-soap-web-client

Objetivos: 1) Exportar uma interface de WebService SOAP para o serviço AssuntoEJB (para todos os métodos, menos configure()); 2) Escrever um cliente SOAP para serviço criado e chamar um ou mais de seus métodos (a) usando um cliente standalone (resolvido); (b) usando um cliente web; 3) Use as ferramentas de outra linguagem que você tenha experiência para escrever um cliente SOAP para o serviço implementado (ex: use ferramentas do Visual Studio, ou WSDL.exe, para gerar proxies a partir do WSDL do serviço implementado em Java, e chame os métodos para listar e obter assuntos.

Este exercício usa como ponto de partida o exercício anterior. O filtro foi alterado para não interceptar urls que terminam em "WebService" para evitar que o acesso ao WSDL peça login. A entidade Assunto foi alterada com algumas anotações (do JAXB) para que a sua conversão em XML ocorra de forma correta. Não houve outras alterações.

O primeiro exercício (em **biblioteca-ws-soap**) poderia ser realizado simplesmente adicionando uma anotação @WebService em **AssuntoEJB**, mas para facilitar a criação de clientes, o serviço deverá ser configurado com um Endpoint Interface, listando os métodos que farão parte da interface (há uma interface no pacote biblioteca.ws). Além disso, é recomendado configurar (na anotação) os nomes do serviço, port e target namespace caso os defaults (gerados a partir dos nomes das classes) não sejam adequados.

O segundo exercício está realizado como aplicação standalone em **assunto-soap-client**. Como exercício proposto está uma versão Web, usando um servlet ou JSF (em **assunto-soap-web-client**).

Como nos exercícios anteriores, há soluções na pasta **solucao/** e exercícios semi-prontos com comentários em **projeto-parcial/**.

O terceiro exercício é opcional mas fortemente recomendado. Escreva um cliente para o serviço Java em outra linguagem de sua preferencia (ex: C#, Python, Objective-C) e chame alguns métodos.

12 Web Service REST

12.1 biblioteca-ws-rest, biblioteca-ws-rest-client, biblioteca-angular-client

Objetivos:

1) Construir e incorporar na aplicação um cliente REST para obter a) as capas dos livros através de ISBN, e b) título e assunto.
2) Exportar serviço de Autor como um WebService REST (permitir listar todos os autores, listar um autor pelo ID, filtrar autores pelo nome, inserir, remover e atualizar).
3) Testar o acesso usando um cliente HTTP (ex: Firefox RESTClient). Use URLs como as abaixo (adapte para os IDs e dados que você tem) pra testar as operações GET:

   ```
   http://localhost:8080/biblioteca-ws-rest/restapi/autor
   http://localhost:8080/biblioteca-ws-rest/restapi/autor/123
   http://localhost:8080/biblioteca-ws-rest/restapi/autor/filterby/D
   ```

   Inclua um cabeçalho Accept com o MIME type correspondente (application/xml ou application/json) para receber um formato ou o outro. Construa um JSON ou XML e envie com PUT (usando um ID) e POST.
4) Construa um outro projeto Java com um cliente REST para testar o acesso aos serviços (CRUD).
5) Construa um cliente em outra linguagem (sugestões: C#, Swift, HTML5/JQuery/Angular) para listar os autores.

Os exercícios 1, 4 e 5 estão resolvidos.

Para o exercício 2 é necessário alterar os arquivos Autor.java, AutorEJB.java e ApplicationConfig.java no projeto biblioteca-ws-rest.

Extras:

a) Melhore a interface da aplicação no exercício 1, de forma a permitir a inserção de autores e editoras; use dados de autor e editora obtidos via ISBN para preencher os dados do livro.
b) Exportar interfaces REST para Editora, Livro, Exemplar, Usuario, Assunto
c) Escrever clientes JavaScript com todas as operações CRUD (ex: use o módulo $resource em Angular)