



**PUCPR**  
GRUPO MARISTA

---

# JavaServer Faces 2.x CRUD

Disciplina de ADS: **Tecnologia para Desenvolvimento Web**



# Sumário – Aplicação Exemplo JSF + CRUD

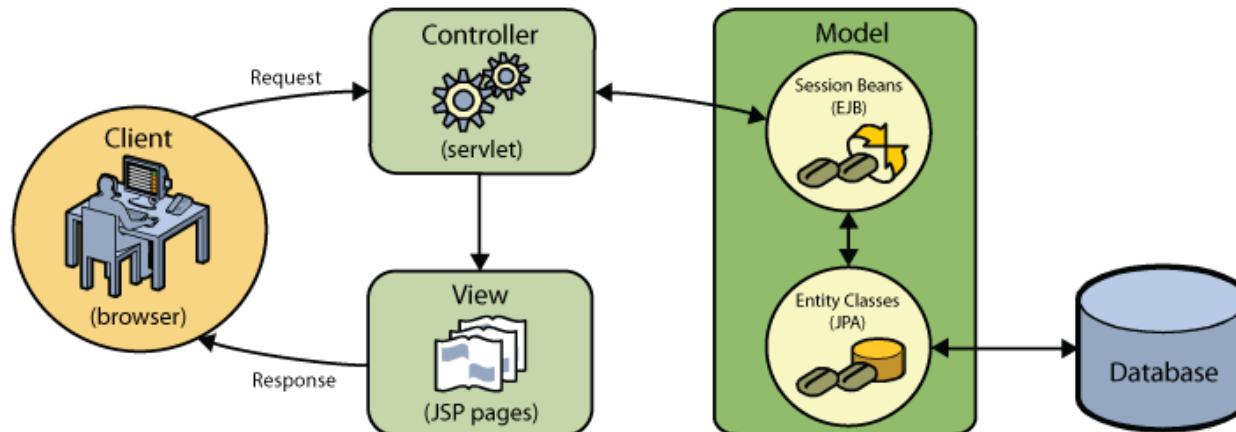
- Apresentação Geral Aplicação Exemplo JSF + CRUD
- Arquitetura de Aplicação EJB + JPA (contextualização)
- Arquitetura da Aplicação Exemplo JSF + CRUD
- Criação da Aplicação Exemplo JSF + CRUD

# Apresentação Geral – Aplicação Exemplo JSF + CRUD

- Tutorial que demonstra como fazer uma aplicação JSF + CRUD (Create-Read-Update-Delete) simples com:
  - **JSF** (JavaServer Faces) – framework de interface com o usuário (UI) para aplicações Java Web
  - **Primefaces** – biblioteca aberta de componentes de interface de usuário (UI) para aplicações Java Web com JSF
  - **JPA** (Java Persistent API) – interface comum para frameworks de persistência de dados; define um meio de **mapeamento objeto-relacional para objetos** Java simples, denominados **entity beans**.
  - **MySQL** – sistema de gerenciamento de banco de dados.
- **Objetivo**
  - A aplicação irá criar, ler, atualizar e excluir (CRUD) de um banco de dados MySQL
- **Requisitos**
  - Ambiente de Desenvolvimento **NetBeans** com Servidor JEE **Payara** com conexão ativa com **MySQL** (ver material **TDW\_Tema1\_2\_JEE-AmbDesenv.pdf**)
  - **Base de dados e tabela** já criadas no MySQL
  - Para consulta: arquivo compactado **JSF\_CRUD.zip**

# Arquitetura de Aplicação EJB + JPA (contextualização)

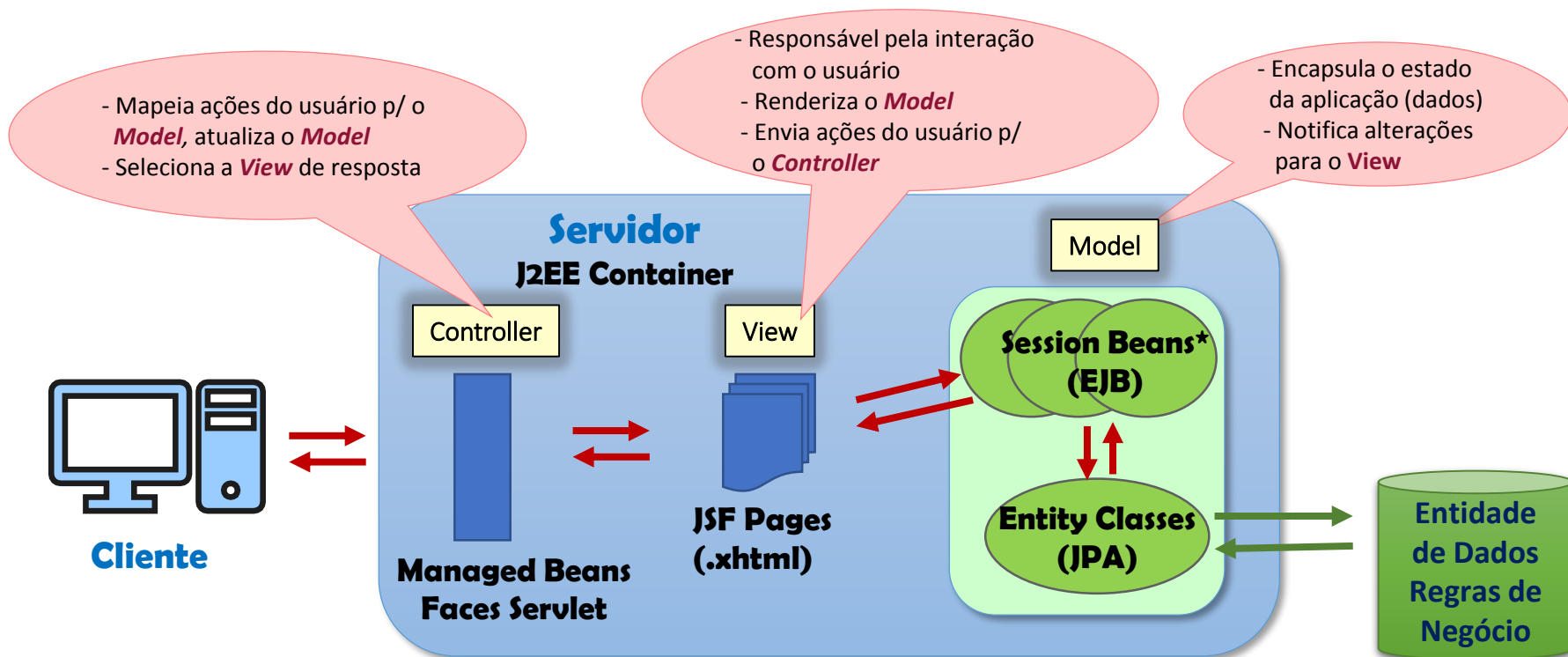
- Para apresentar a nossa arquitetura da aplicação exemplo deste tutorial, resgatamos a arquitetura de acesso a BD com Sessions Beans (**EJB**) e Entity Classes (**JPA**), em aplicação **JSP** (precursora da **JSF**)
  - **Session Beans (EJB)** – intermedia o acesso à classe Entity, interagindo como Controller
  - **Entity Classes (JPA)** – representa o estado persistido (tabela em BD relacional) da aplicação



fonte: <https://netbeans.org/kb/docs/javaee/ecommerce/entity-session.html>

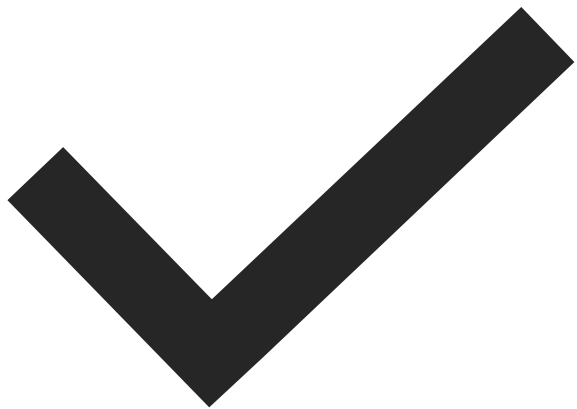
# Arquitetura da Aplicação Exemplo JSF + CRUD

- Nossa arquitetura de aplicação com Sessions Beans (EJB) e Entity Classes (JPA), em aplicação JSF promove o **Modelo MVC** (Model – View – Controller), em que a aplicação é decomposta em componentes lógicos que podem ser arquitetados mais facilmente, melhorando a flexibilidade e reutilização de código



\* Session Beans - cria 3 classes para intermediar o acesso ao JPA (tabela em banco de dados):

- ✓ **AbstractFacade:** classe abstrata (classe modelo que não instancia) que contém métodos abstratos para todas as classes de entidade.
- ✓ **BooksFacade:** classe onde os métodos encontrados no AbstractFacade devem ser implementados relacionados à classe de entidade Book.
- ✓ **BooksFacadeLocal:** interface contém todos os serviços fornecidos pelo Session Bean como métodos.



Vamos começar o  
desenvolvimento  
da Aplicação  
Exemplo JSF +  
CRUD!

# 1 - Base de Dados CRUD

- No gerenciador do **MySQL**, o **PHPMyAdmin** (<http://localhost/phpmyadmin>), crie a **base de dados crud** e a **tabela book**, como indicado no código SQL abaixo (realize os passos um a um no **phpMyAdmin** do **MySQL**, ou execute o código SQL correspondente no **phpMyAdmin** – neste último caso, obtenha o arquivo **crud.sql** no material de apoio

## Dados para conexão:

Base de Dados = **crud**  
Usuário : **crud**  
Password: **1234**

SQL para  
criar e  
povoar  
tabela  
**book**

Tabela Book

```
SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO";
SET AUTOCOMMIT = 0;
START TRANSACTION;
SET time_zone = "+00:00";

CREATE TABLE `book` (
  `ID` int(11) NOT NULL,
  `name` varchar(254) NOT NULL,
  `author` varchar(254) NOT NULL,
  `category` varchar(254) NOT NULL,
  `year` int(11) DEFAULT NULL,
  `price` float NOT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1;

INSERT INTO `book` (`ID`, `name`, `author`, `category`, `year`, `price`) VALUES
(1, 'Livro 1', 'Autor 1', 'Categoria 1', 2017, 10),
(2, 'Livro 2', 'Autor 2', 'Categoria 2', 2018, 20),
(4, 'Livro 3', 'Autor 3', 'Categoria 3', 2016, 20.5);

ALTER TABLE `book`
  ADD PRIMARY KEY (`ID`);

ALTER TABLE `book`
  MODIFY `ID` int(11) NOT NULL AUTO_INCREMENT, AUTO_INCREMENT=5;
COMMIT;
```

# 1 - Base de Dados CRUD

- No gerenciador do **MySQL**, o **PHPMyAdmin** (<http://localhost/phpmyadmin>), a base de dados **crud** e a tabela **book**, com 6 colunas, deve ter a configuração indicada na figura abaixo

The screenshot shows the PHPMyAdmin interface. On the left, the 'Base CRUD' label points to the 'crud' database in the left sidebar. On the right, the 'Tabela book' label points to the 'book' table in the top navigation bar. The main area displays the table structure and data for 'book'.

Base CRUD

Tabela book

Servidor: 127.0.0.1 » Base de Dados: crud » Tabela: book

Procurar Estrutura SQL Pesquisar Inserir Exportar Importar

✓ A mostrar registos de 0 - 3 (4 total, A consulta demorou 0,0000 segundos.)

`SELECT * FROM `book``

☐ Mostrar tudo | Número de registos: 25 | Filtrar registos: Pesquisar esta tabela

+ Opções

	ID	name	author	category	year	price
<input type="checkbox"/>	1	Livro 1	Autor 1	Categoria 1	2017	10
<input type="checkbox"/>	2	Livro 2	Autor 2	Categoria 2	2018	20
<input type="checkbox"/>	4	Livro 3	Autor 3	Categoria 3	2016	20.5

☐ Marcar todos Com os seleccionados:



## 2 - Configurando a conexão com a BD

Na console de administração do **Servidor Payara**, indicaremos ao servidor como acessar o BD crud no MySQL, criando uma **Connection Pool** e seu respectivo **JNDI**

- Vá para o **console de administração do Payara** em **http://localhost: 4848**.
- Depois, vá para **Resources → JDBC → JDBC Connection Pools**. Clique no botão **New**.

The screenshot displays the Payara Server administration console interface. At the top, the user is logged in as 'admin' on 'localhost'. The left sidebar shows a navigation tree with 'Resources' expanded and 'JDBC Connection Pools' selected, highlighting the 'crud' pool. The main panel is titled 'Edit JDBC Connection Pool' and contains the following configuration fields:

- General Settings** tab is selected.
- Pool Name:** crud
- Resource Type:** java.sql.Driver (selected from a dropdown menu)
- Datasource Classname:** (empty text field)
- Driver Classname:** com.mysql.jdbc.Driver
- Ping:** Enabled (checkbox)

Red arrows highlight the 'General' tab, the 'Resource Type' dropdown, and the 'Driver Classname' field.

## 2 - Configurando a conexão com a BD (continuação)

- Ainda em **Resources** → **JDBC** → **JDBC Connection Pools**, clique na aba **Additional Properties** e preencha os valores para que o **Payara** conecte-se como **MySQL**

User: admin Domain: domain1 Server: localhost

Home About... Help Online Help Enable Asadmin Recorder

payara server 5

Server (Admin Server)  
Deployment Groups  
Instances  
Nodes  
Clusters (Deprecated)  
Applications  
Lifecycle Modules  
Monitoring Data  
Resources  
Concurrent Resources  
Connectors  
JDBC  
JDBC Resources  
JDBC Connection Pools  
crud  
DerbyPool  
H2Pool  
SamplePool  
\_\_TimerPool  
JMS Resources  
JNDI

General Advanced **Additional Properties**

**Edit JDBC Connection Pool Properties** Save Cancel

Modify properties of an existing JDBC connection pool.

Pool Name: crud

User: crud  
Password: 1234  
URL: jdbc:mysql://localhost:3306/crud?zeroDateTimeBehavior=convertToNull

Additional Properties (3)

Add Property Delete Properties

Select	Name	Value	Description
<input type="checkbox"/>	password	1234	
<input type="checkbox"/>	user	crud	
<input type="checkbox"/>	URL	jdbc:mysql://localhost:3306/crud?zeroDateTimeBeh	

## 2 - Configurando a conexão com a BD (continuação)

- Ainda em **Resources** → **JDBC** → **JDBC Connection Pools**, volte para a aba **General** e acione o botão **Ping**, que deve ser bem sucedido

The screenshot shows the Payara Server administration console. The top navigation bar includes links for Home, About, Help, and Online Help. The left sidebar shows the navigation tree with 'Resources' expanded, and 'JDBC' selected. Under 'JDBC', 'JDBC Connection Pools' is selected, and the 'crud' pool is highlighted. The main content area shows the 'Edit JDBC Connection Pool' page for the 'crud' pool. The 'General' tab is active, and the 'Ping' button is highlighted. A 'Ping Succeeded' message is displayed in a green box. The 'General Settings' section shows the 'Pool Name' as 'crud' and the 'Resource Type' as 'java.sql.Driver'.

User: admin Domain: domain1 Server: localhost

Home About... Help Online Help Enable Asadmin Recorder

payara server 5

Applications  
Lifecycle Modules  
Monitoring Data  
Resources  
Concurrent Resources  
Connectors  
JDBC  
JDBC Resources  
Jcrud  
jdbc/\_\_\_TimerPool  
jdbc/\_\_\_default  
jdbc/\_\_\_derby  
jdbc/sample  
JDBC Connection Pools  
crud  
DerbyPool  
H2Pool  
SamplePool  
\_\_\_TimerPool  
JMS Resources  
JNDI

General Advanced Additional Properties

Ping Succeeded

Edit JDBC Connection Pool

Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.

Load Defaults Flush Ping Save Cancel

\* Indicates required field

General Settings

Pool Name: crud

Resource Type: java.sql.Driver

Datasource Classname:

## 2 - Configurando a conexão com a BD (continuação)

- Depois, vá para **Resources** → **JDBC** → **JDBC Resources**.
- Crie o recurso JBDC **Jcrud** para o Pool **crud**. Após, acione o botão **Save**

User: admin Domain: domain1 Server: localhost Home About... Help Online Help Enable Asadmin Recorder

payara server 5

Deployment Groups  
Instances  
Nodes  
Clusters (Deprecated)  
Applications  
Lifecycle Modules  
Monitoring Data  
Resources  
Concurrent Resources  
Connectors  
JDBC  
JDBC Resources  
Jcrud  
jdbc/\_TimerPool  
jdbc/\_default  
jdbc/\_derby  
jdbc/sample

### Edit JDBC Resource

Edit an existing JDBC data source.

Load Defaults

JNDI Name: Jcrud

Pool Name: crud

Use the [JDBC Connection Pools](#) page to create new pools

Deployment Order: 100

Specifies the loading order of the resource at server startup. Lower numbers are loaded first.

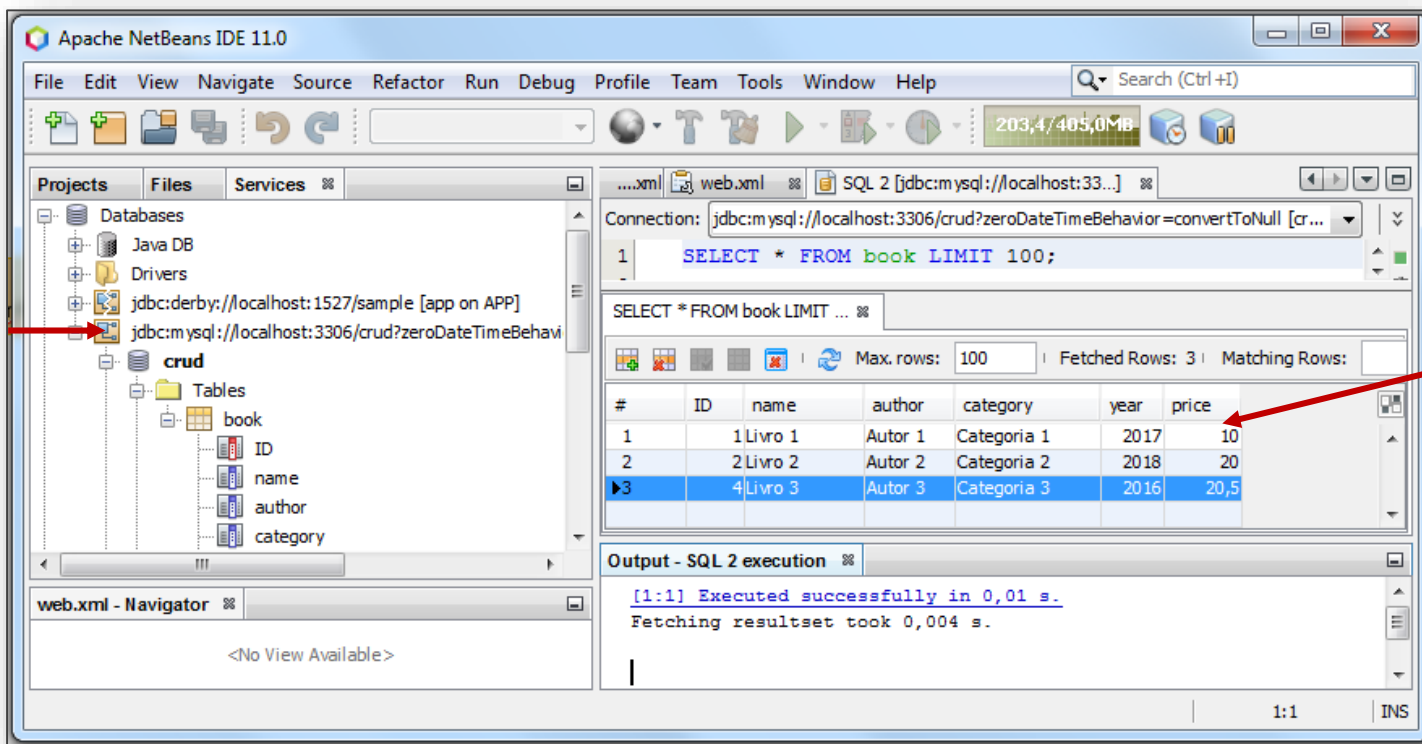
Description:

Status: ☒ Enabled

Save Cancel

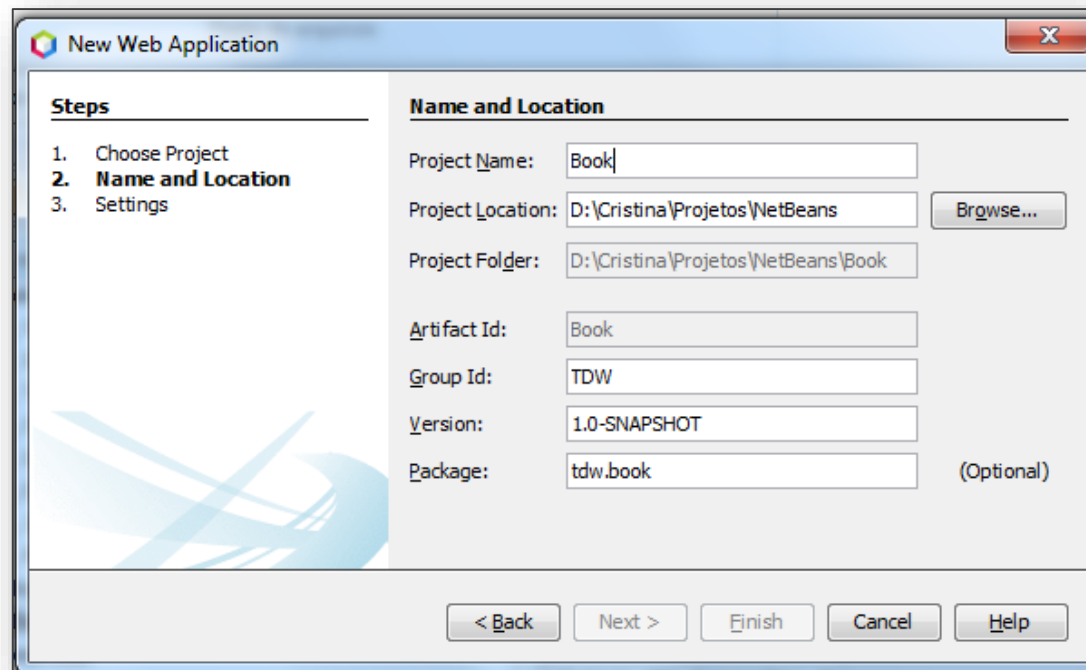
## 2 - Configurando a conexão com a BD (continuação)

- Siga os procedimentos indicados para configuração do ambiente de desenvolvimento e execute a conexão com o banco de dados criado anteriormente.
- Quando a conexão estiver **Ok**, será possível acessar os dados no MySQL, como indicado na figura abaixo.



# 3 - Criando o Projeto Book no NetBeans

- Na janela **Projects**, clique em **New Project**
- Selecione **Categories > Java with Maven**
- Selecione **Project > Web Application**
- Clique **Next**
- No assistente, preencha os campos como indicado:



The screenshot shows the 'New Web Application' dialog box in NetBeans. The 'Steps' panel on the left indicates the current step is '2. Name and Location'. The 'Name and Location' panel on the right contains the following fields:

Name and Location	
Project Name:	Book
Project Location:	D:\Cristina\Projetos\NetBeans
Project Folder:	D:\Cristina\Projetos\NetBeans\Book
Artifact Id:	Book
Group Id:	TDW
Version:	1.0-SNAPSHOT
Package:	tdw.book

Buttons at the bottom: < Back, Next >, Finish, Cancel, Help. A 'Browse...' button is next to the Project Location field. The Package field is marked as '(Optional)'.

## 4 - Criando nossas classes Entity

- Selecione o projeto **Book** e com botão direito do mouse e clique em **New File**
- Selecione **Categories > Persistence**
- Selecione **File Types > Entity Classes from DataBase**
- Adicione a tabela de livros às tabelas selecionadas e clique **Next**

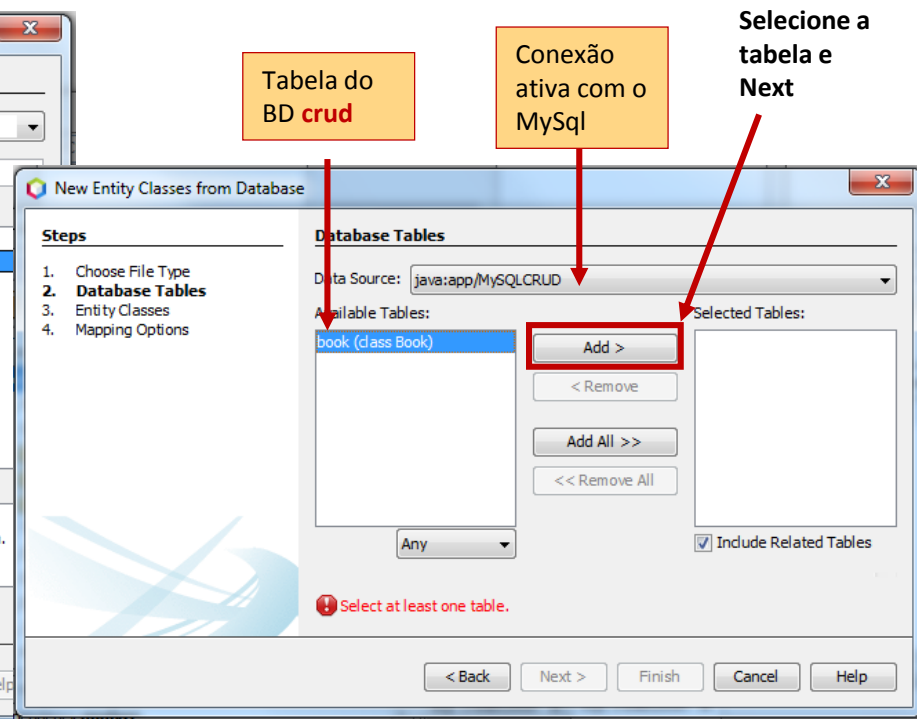
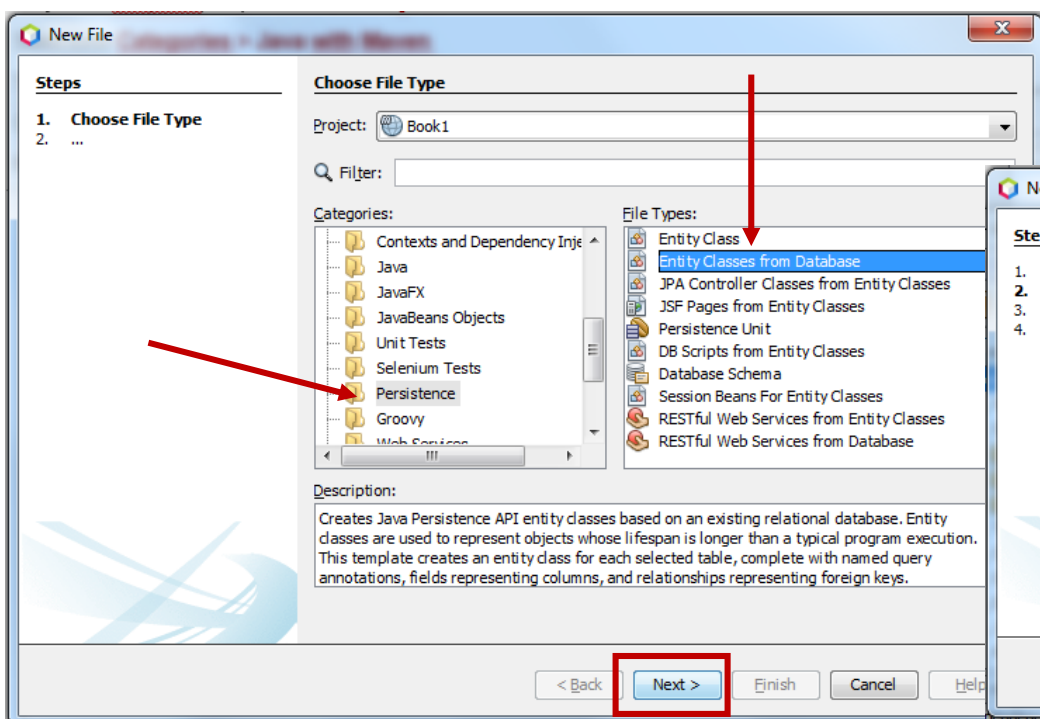


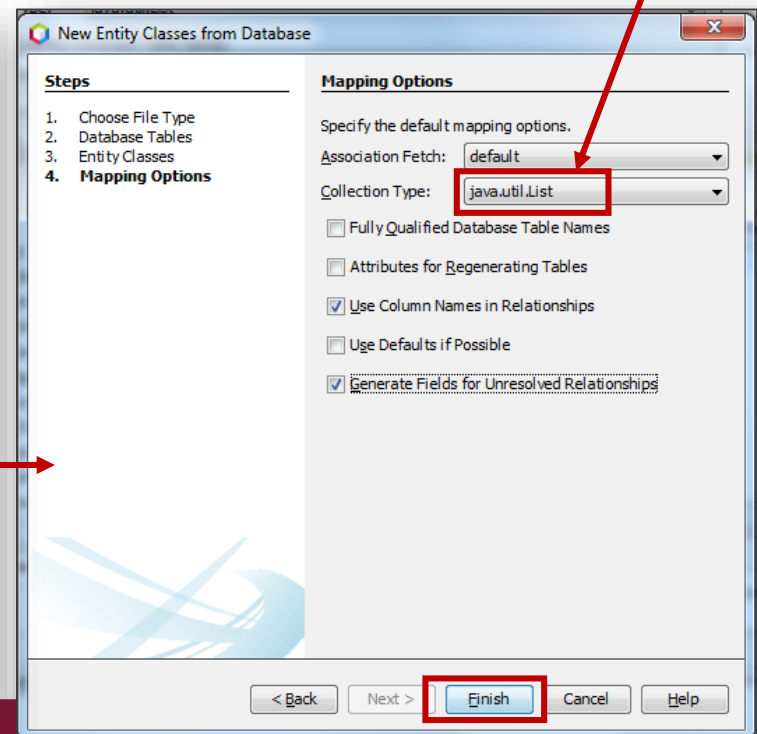
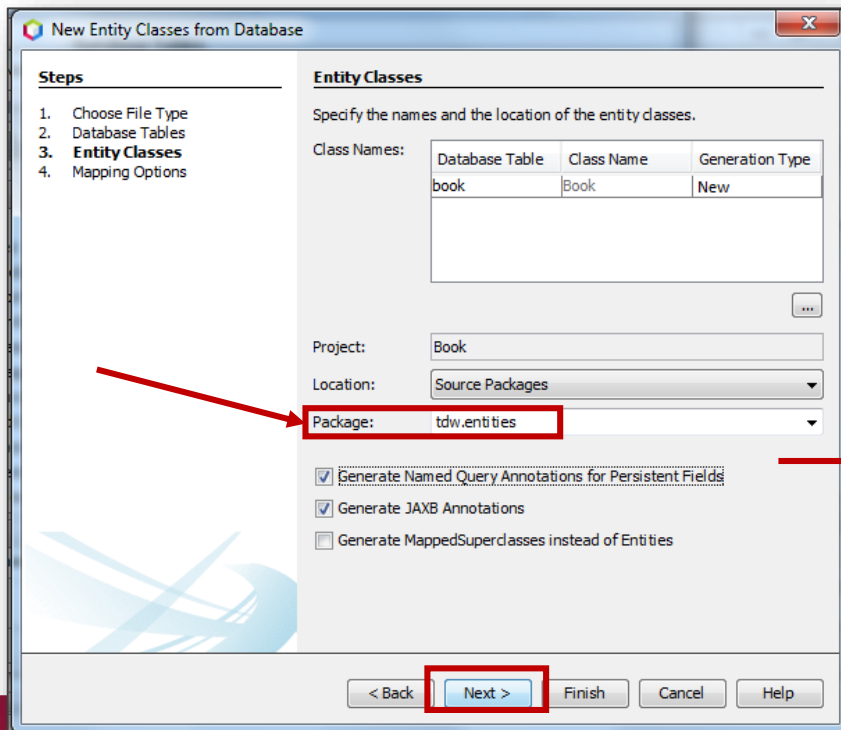
Tabela do  
BD crud

Conexão  
ativa com o  
MySQL

Selecione a  
tabela e  
Next

## 4 - Criando nossas classes Entity (continuação)

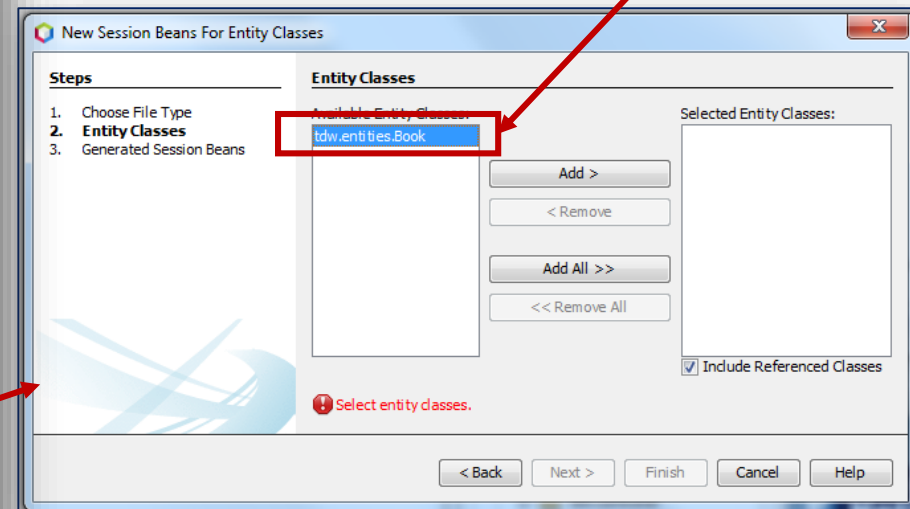
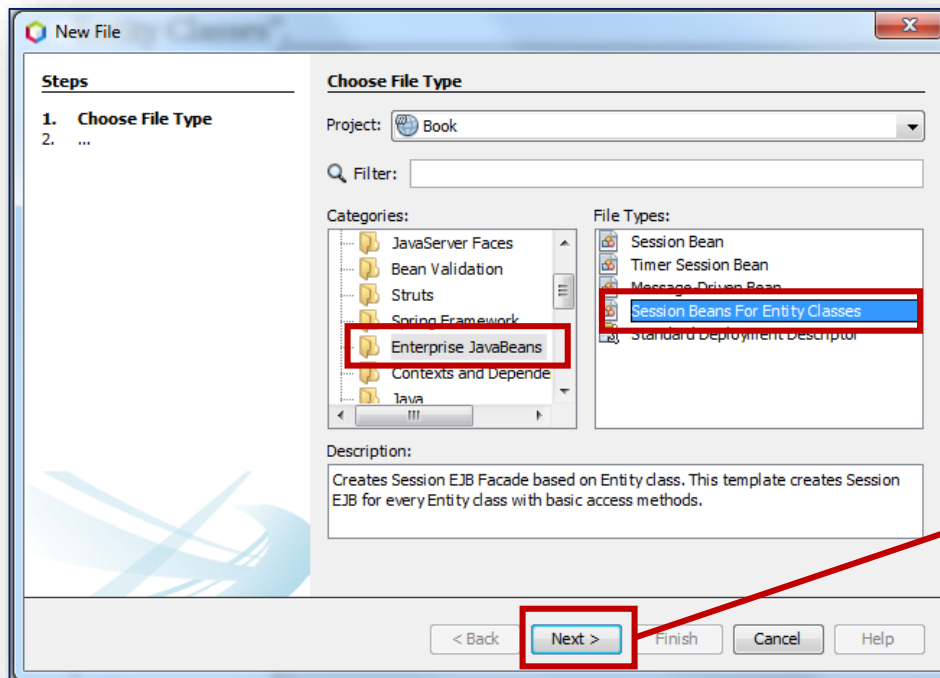
- Vamos gerar nossa classe em um novo pacote chamado “**entities**”
- Defina a **Association Fetch** como **default** e o **Collection Type** (tipo de coleção) como **java.util.List** – isso descreve a maneira como devemos recuperar nossas entidades a partir do banco de dados
- Após o **Finish**, geramos com sucesso nossa **entity class Book**: ela contém consultas nomeadas, variáveis de entidade com suas restrições, um construtor, getters e setters e algumas poucas funções.





## 5 - Criando nosso Session Beans (EJB)

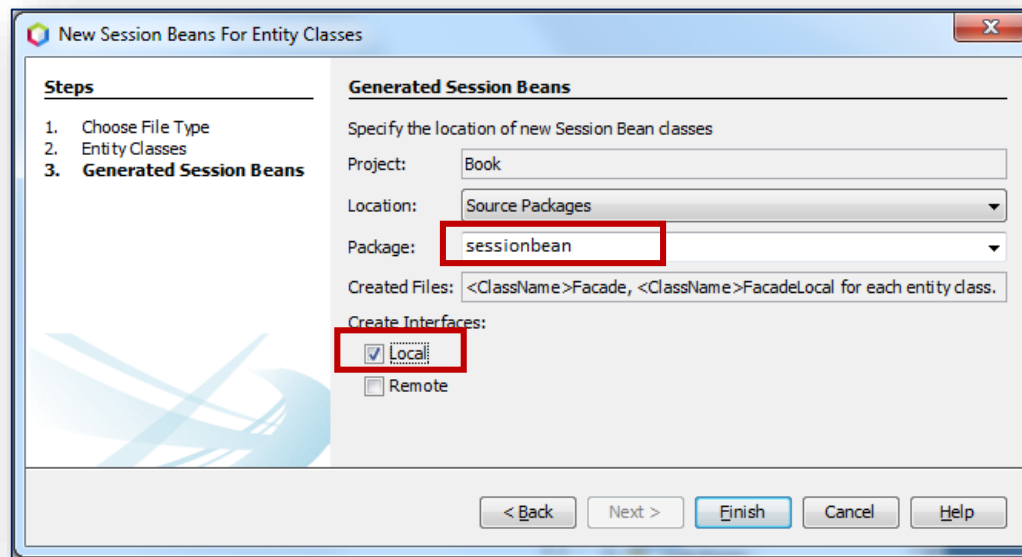
- Um **Session Bean** é um componente do lado do servidor que fornece serviços aos clientes.
- Ele vai fornecer as diferentes etapas de CRUD para um fluxo de trabalho da nossa aplicação:
- Para criar nosso **Session Bean**, crie um novo arquivo e escolha **Session Beans For Entity Classes**..



Escolha a classe de entidade correspondente (**tdw.entities.Book**) e vá para a próxima etapa

## 5 - Criando nosso Session Beans (EJB) (continuação)

- Vamos criar nosso Session Bean\* em um novo pacote chamado **sessionbean**.
- Definir a interface para **local**, pois o cliente é implantado na mesma máquina virtual, ou seja, nossa aplicação Web está no mesmo servidor de aplicações que o Session Bean

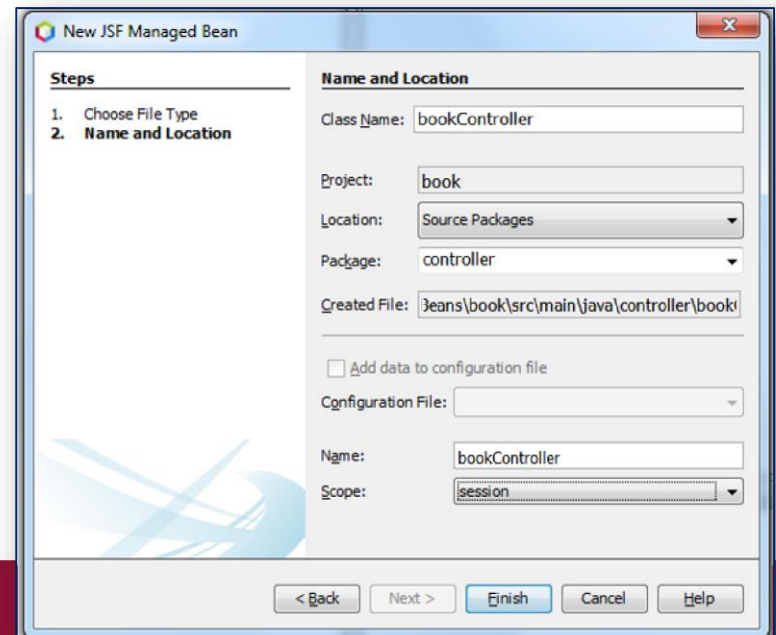
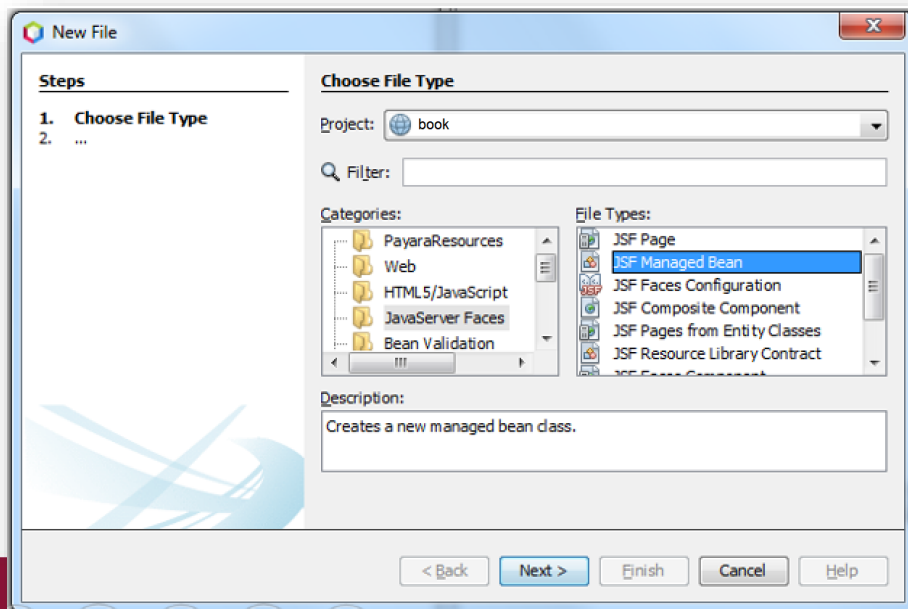


\* **Session Beans** - cria 3 classes para intermediar o acesso ao JPA (tabela em banco de dados):

- ✓ **AbstractFacade**: classe abstrata (classe modelo que não instancia) que contém métodos abstratos para todas as classes de entidade.
- ✓ **BooksFacade**: classe onde os métodos encontrados no AbstractFacade devem ser implementados relacionados à classe de entidade Book.
- ✓ **BooksFacadeLocal**: interface contém todos os serviços fornecidos pelo Session Bean como métodos.

## 6 - Criando nosso Managed Bean

- Na aba **Projects**, clique com o botão direito do mouse no projeto **Book** e selecione **New > JSF Managed Bean**. (Se o **JSF Managed Bean** não estiver listado, selecione **Others**. Em seguida, selecione a opção **JSF Managed Bean** na categoria **JavaServer Faces**. Clique em **Próximo**).
- No assistente, informe o seguinte:
  - **Nome da Classe:** bookController
  - **Pacote:** tdw.controller
  - **Nome:** bookController
  - **Escopo:** Session (os objetos e atributos do **managed bean** são mantidos durante a sessão do navegador)



## 6 - Criando nosso Managed Bean (continuação)

```
package tdw.controller;

import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
import java.io.Serializable;
import java.util.List;
import javax.ejb.EJB;
import tdw.entities.Book;
import tdw.sessionbean.BookFacadeLocal;

@Named(value = "bookController")
@SessionScoped
public class bookController implements Serializable {

    @EJB
    private BookFacadeLocal booksFacade;

    private Book selectedBook;
    private final Book book = new Book();
    private String name;
    private String author;
    private Integer year;
    private String category;
    private float price;

    public bookController() { }

    public Book getSelectedBook() {return selectedBook;

    public void setSelectedBook(Book selectedBook) {
        this.selectedBook = selectedBook;    }
```

```
public String getName() { return name; }
public void setName(String name) { this.name = name; }
public String getAuthor() { return author; }
public void setAuthor(String author) { this.author = author; }
public Integer getYear() { return year; }
public void setYear(Integer year) { this.year = year; }
public String getCategory() { return category; }
public void setCategory(String category) {this.category = category; }
public float getPrice() { return price; }
public void setPrice(float price) { this.price = price; }
```

```
public void emptyVariables() {
    this.author = "";
    this.category = "";
    this.name = "";
    this.price = 0;
    this.year = 0;
}
```

```
public String createBook() {
    this.book.setAuthor(this.author);
    this.book.setCategory(this.category);
    this.book.setName(this.name);
    this.book.setPrice(this.price);
    this.book.setYear(this.year);
    this.booksFacade.create(this.book);
    this.emptyVariables();
    return "index.xhtml?faces-redirect=true";
}
```

```
public List<Book> getAllBooks() {
    return this.booksFacade.findAll();
}
```

```
public String updateBook(Book book) {
    this.booksFacade.edit(this.selectedBook);
    return "index.xhtml?faces-redirect=true";
}
```

```
public String deleteBook(Book book) {
    this.booksFacade.remove(book);
    return "index.xhtml?faces-redirect=true";
}
```

```
}
```

## 7 - Criando e construindo nossa View

- **Criação:** com o projeto Book selecionado, clique o botão direito do mouse e crie uma nova página **JSF** chamada **index.xhtml**
  - Importante: exclua o arquivo **index.html** existente.
- **Construção:** após, vamos construir nossa página de **index.xhtml**, criando:
  - um **formulário** para adicionar dados à nossa tabela **book** e
  - uma **tabela de dados** para exibir os registros da nossa tabela de **book**.
  - a própria tabela também nos permitirá **atualizar** e **excluir** um livro escolhido.

# 7 - Construindo nossa View

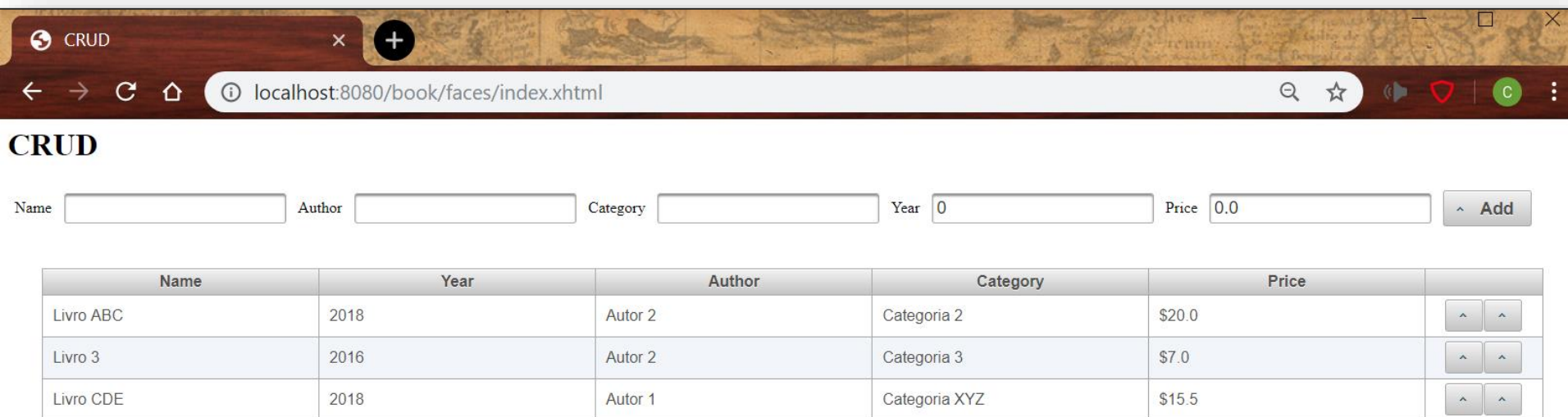
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:p="http://primefaces.org/ui"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
<h:head>
  <title>CRUD</title>
</h:head>
<h:body>
  <h1>CRUD</h1>
  <h:form>
    <h:panelGrid columns="12" cellpadding="5">
      <h:outputText value="Name " />
      <p:inputText value="#{bookController.name}" />
      <h:outputText value="Author " />
      <p:inputText value="#{bookController.author}" />
      <h:outputText value="Category " />
      <p:inputText value="#{bookController.category}" />
      <h:outputText value="Year " />
      <p:inputText value="#{bookController.year}" />
      <h:outputText value="Price " />
      <p:inputText value="#{bookController.price}" />
      <p:commandButton value="Add" icon="fa fa-fw fa-plus" action="#{bookController.createBook()}" />
    </h:panelGrid>
  </h:form>
  <h:form id="form">
    <p:dataTable value="#{bookController.getAllBooks()}" var="book" style="margin: 2em;" rowKey="#{book.id}">
      <p:column headerText="Name">
        <h:outputText value="#{book.name}" />
      </p:column>
      <p:column headerText="Year">
        <h:outputText value="#{book.year}" />
      </p:column>
      <p:column headerText="Author">
        <h:outputText value="#{book.author}" />
      </p:column>
      <p:column headerText="Category">
        <h:outputText value="#{book.category}" />
      </p:column>
      <p:column headerText="Price">
        <h:outputText value="$#{book.price}" />
      </p:column>
    </p:dataTable>
  </h:form>
</h:body>
</html>
```

## 7 - Construindo nossa View (continuação)

```
<p:column style="width:100px;text-align: center">
    <p:commandButton icon="fa-pencil" update=":form:bookEdit" oncomplete="PF('editDialog').show()">
        <f:setPropertyActionListener value="#{book}" target="#{bookController.selectedBook}"/>
    </p:commandButton>
    <p:commandButton action="#{bookController.deleteBook(book)}" icon="fa-trash"></p:commandButton>
</p:column>
</p:dataTable>
<p:dialog header="Edit Book" widgetVar="editDialog" modal="true" showEffect="fade" hideEffect="fade" resizable="false">
    <p:outputPanel id="bookEdit" style="text-align:center;">
        <p:panelGrid columns="2" rendered="#{not empty bookController.selectedBook}" columnClasses="label,value">
            <h:outputText value="Name: " />
            <p:inputText value="#{bookController.selectedBook.name}" />
            <h:outputText value="Category: " />
            <p:inputText value="#{bookController.selectedBook.category}" />
            <h:outputText value="Author: " />
            <p:inputText value="#{bookController.selectedBook.author}"/>
            <h:outputText value="Price: $" />
            <p:inputText value="#{bookController.selectedBook.price}" />
            <h:outputText value="Year: " />
            <p:inputText value="#{bookController.selectedBook.year}" />
        </p:panelGrid>
        <p:commandButton value="Update" icon="fa-pencil" action="#{bookController.updateBook(book)}"/>
    </p:outputPanel>
</p:dialog>
</h:form>
</h:body>
</html>
```

## 8 - Construindo a aplicação JSF + CRUD

- Com o projeto book selecionado, realize o **Clean and build**
- Em seguida, realize o **Run** para implantar (deploy) a aplicação no servidor Payara e verificar seu funcionamento no navegador
- A aplicação ficará semelhante ao apresentado na figura abaixo



CRUD

Name  Author  Category  Year  Price

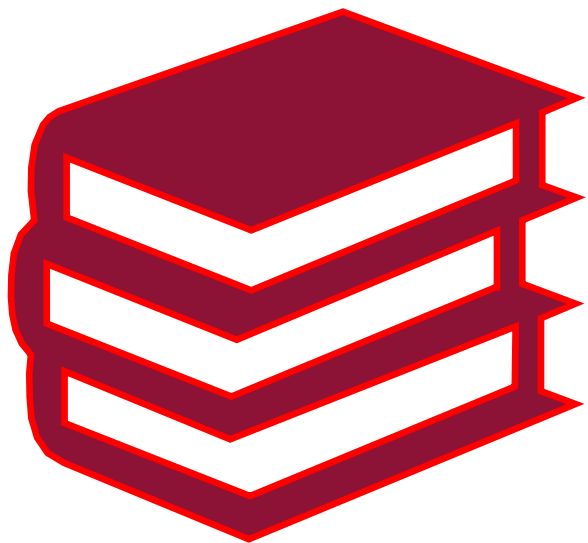
Name	Year	Author	Category	Price		
Livro ABC	2018	Autor 2	Categoria 2	\$20.0	⬆	⬆
Livro 3	2016	Autor 2	Categoria 3	\$7.0	⬆	⬆
Livro CDE	2018	Autor 1	Categoria XYZ	\$15.5	⬆	⬆



**Ao trabalho!**

Agora, execute os procedimentos descritos neste documento para construir seu exemplo de aplicação JSF CRUD com MySQL!

Você poderá ver a construção e execução completas do Projeto, em um [vídeo tutorial](#) para esta tarefa: verifique nos links de apoio



# Glossário

# Glossário

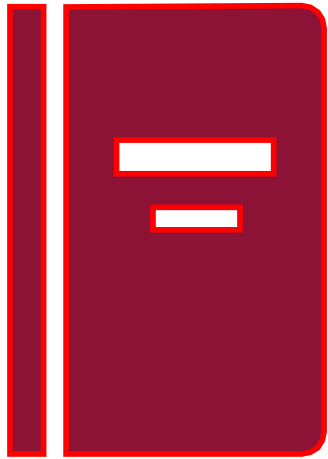
- **Connection Pool:** ou “piscina de conexões”, é uma camada entre o cliente e o BD (dentro do servidor de aplicações) que intermedia as conexões entre eles para permitir vários acessos simultâneos de usuários às páginas Web e ao BD, com reaproveitamento de conexões abertas. Em aplicações Java:
  - **Cliente** = um EJB, um Servlet, ou uma classe Java qualquer
  - **BD** = conexão com o driver JDBC para algum servidor de banco de dados.
  - <https://www.devmedia.com.br/connection-pool/5869>
  - [https://pt.wikipedia.org/wiki/Pool\\_de\\_conex%C3%B5es](https://pt.wikipedia.org/wiki/Pool_de_conex%C3%B5es)
- **JNDI** (Java Naming and Directory Interface): é uma API utilizada em aplicações que acessam recursos externos, permitindo assim obter esses recursos através de um nome.
  - <https://pt.wikipedia.org/wiki/JNDI>
  - <https://docs.oracle.com/javase/tutorial/jndi/overview/index.html>

# Glossário

- **Entity Beans (EJB)**: é um tipo de Enterprise JavaBean, um componente JEE do lado servidor, que representa dado persistente mantido em um banco de dados.
  - <https://javaee.github.io/tutorial/ejb-basicexamples.html#GJIRB>
  - <https://www.devmedia.com.br/apresentando-ejb-entity-bean-cmp/7049>
- **JSF** (JavaServer Faces): framework de interface com o usuário (UI) para aplicações Java Web
- **Primefaces**: biblioteca aberta de componentes de interface de usuário (UI) para aplicações Java Web com JSF
  - [https://netbeans.org/kb/docs/javaee/maven-primefaces-screencast\\_pt\\_BR.html](https://netbeans.org/kb/docs/javaee/maven-primefaces-screencast_pt_BR.html)
  - <https://www.primefaces.org/>
  - <https://www.devmedia.com.br/introducao-ao-primefaces/33139>

# Glossário

- **JPA** (Java Persistent API): interface comum para frameworks de persistência de dados; define um meio de mapeamento objeto-relacional para objetos Java simples, denominados entity beans.
  - <https://netbeans.org/kb/docs/javaee/ecommerce/entity-session.html>
  - <https://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html>
  - <https://www.devmedia.com.br/introducao-a-jpa-java-persistence-api/28173>
  - <https://www.devmedia.com.br/java-persistence-api-jpa-primeiros-passos/30511>
- **MySQL**: sistema de gerenciamento de banco de dados
  - <https://www.mysql.com/>



# Referências

# Referências

- Material baseado no tutorial NetBeans em
  - Creating a J2EE CRUD app in 10 Steps with JSF + Primefaces + JPA + MySQL
    - <https://hackernoon.com/creating-a-j2ee-crud-app-in-10-steps-with-jsf-primefaces-jpa-mysql-39a1421b8845>



**PUCPR**

GRUPO MARISTA

**PROFESSOR CONTEUDISTA**

**Cristina Verçosa Pérez Barrios de Souza, Profa. Dra.**

© PUCPR. Todos os direitos reservados.

Nenhum texto pode ser reproduzido sem prévia autorização.