



Transcrição

Seja bem-vindo ao curso de CDI da Alura.

Neste curso vamos ver alguns recursos bem peculiares e alguns um pouco mais avançados.

do CDI.

Na plataforma da Alura, temos um curso introdutório de JSF (<https://cursos.alura.com.br/course/jsf-cdi>) no qual abordamos como utilizar o CDI.

Nós iremos utilizar como base o projeto do JSF com CDI. Nosso objetivo é configurar o projeto para utilizar o CDI e, em seguida, criar uma biblioteca construída por nós, e que poderá ser reutilizada em outros projetos que utilizem a plataforma Java EE.

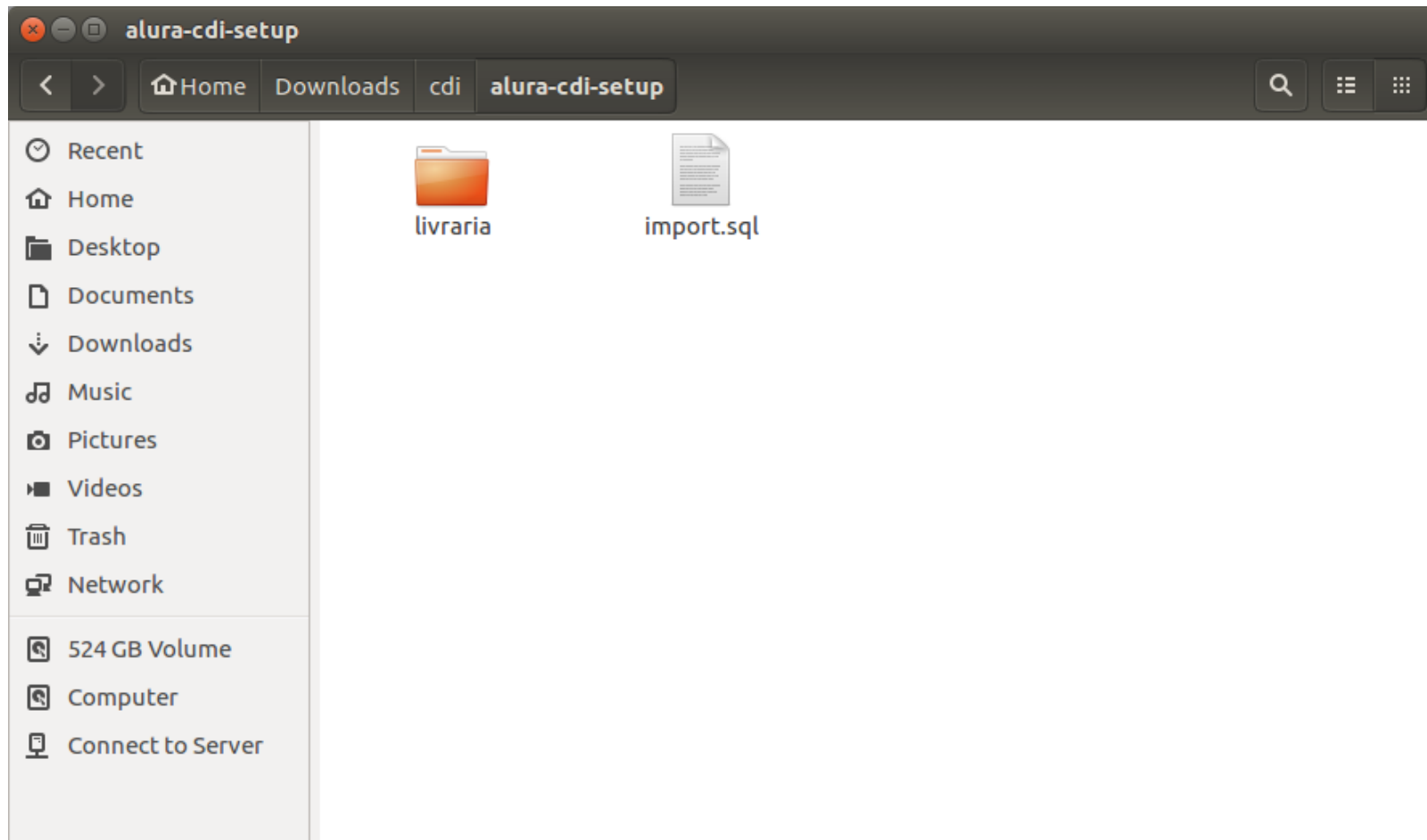
Apesar de utilizar como base o projeto de um outro curso, o projeto atual está um pouco diferente. Agora estamos utilizando o [Maven](https://maven.apache.org/) (<https://maven.apache.org/>) para fazer o gerenciamento do `_build_`. Em vez de adicionar de forma manual todos os `jars` dentro do `_classpath_`, vamos utilizar o Maven para baixar os `jars` e adicioná-los dentro do `_classpath_`.

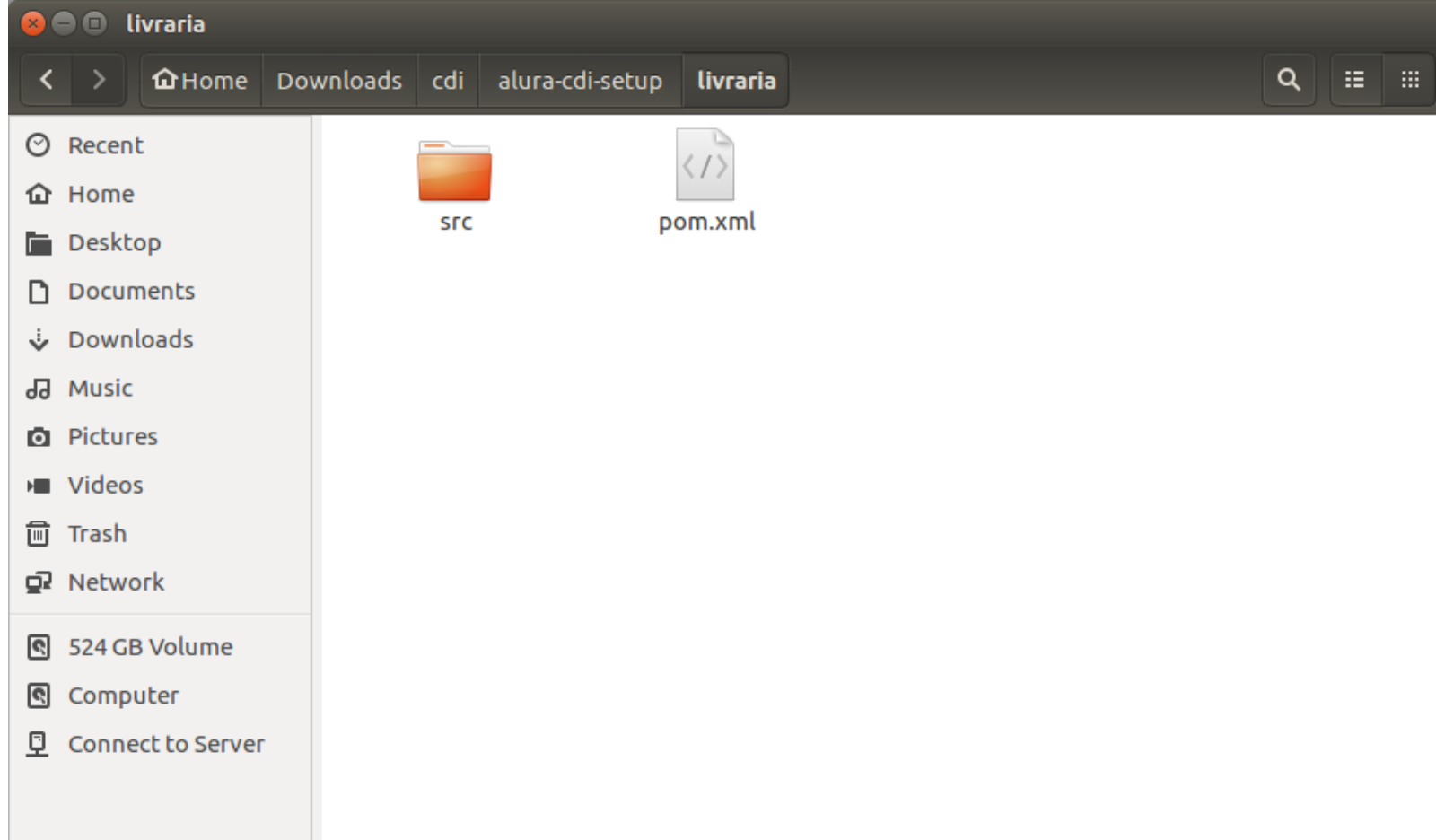
Se você não conhece o Maven, sugerimos que você faça antes nosso [curso de Maven](https://cursos.alura.com.br/course/maven-build-do-zero-a-web) (<https://cursos.alura.com.br/course/maven-build-do-zero-a-web>). E voltar depois para este curso, já com um entendimento do que é o Maven e o que a ferramenta nos possibilita fazer.

A mesma recomendação serve para o JSF. Vamos falar de algumas coisas específicas do JSF, utilizando o CDI, mas é interessante que você já conheça um pouco do JSF.

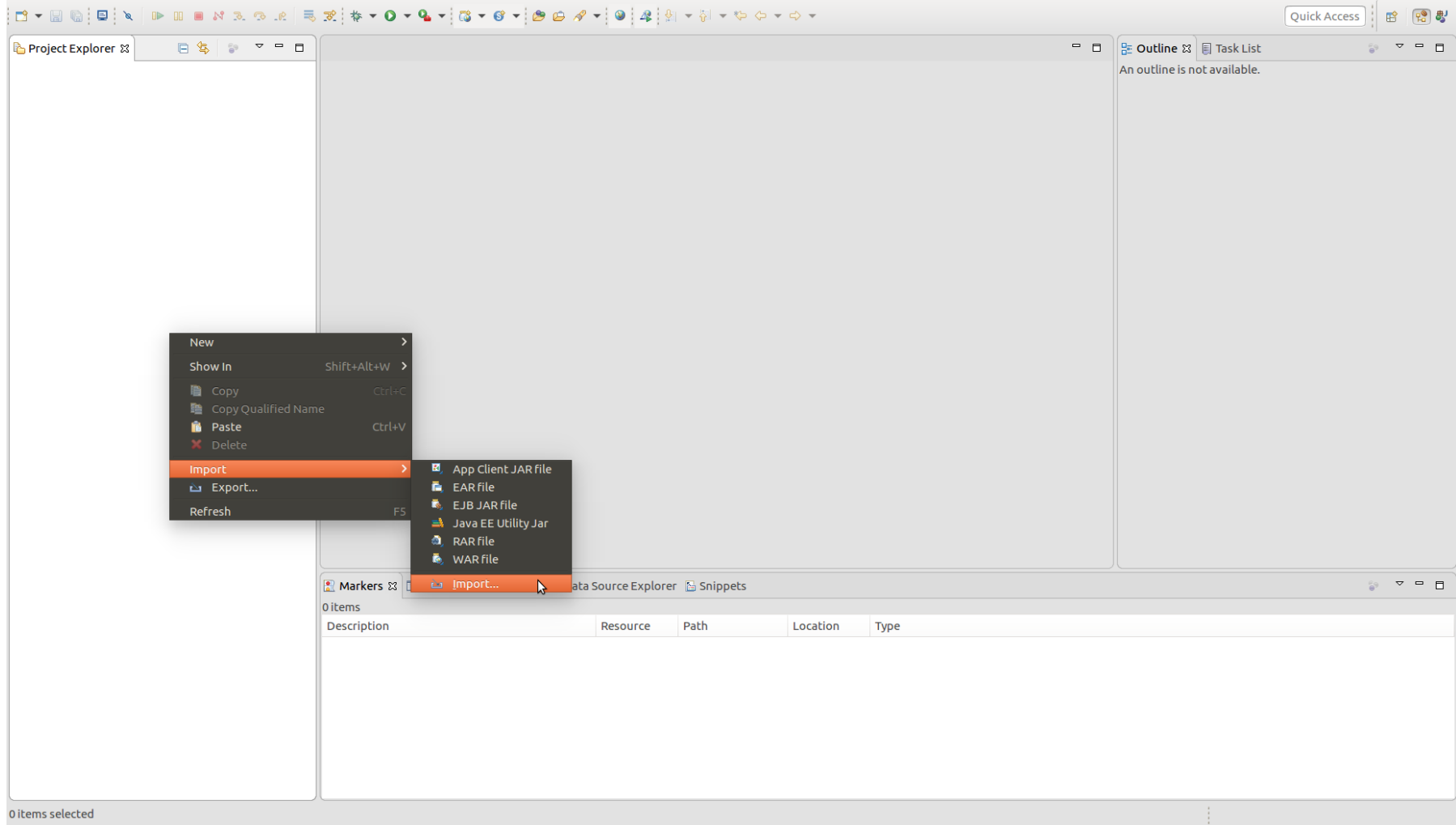
Vamos começar importando o nosso projeto. Como citado anteriormente, vamos utilizar um projeto Maven.

Descompactamos o projeto em algum diretório do sistema, e temos o diretório `livraria`, que dentro dele possui o arquivo `pom.xml` do Maven. Agora podemos simplesmente importar o projeto no Eclipse.

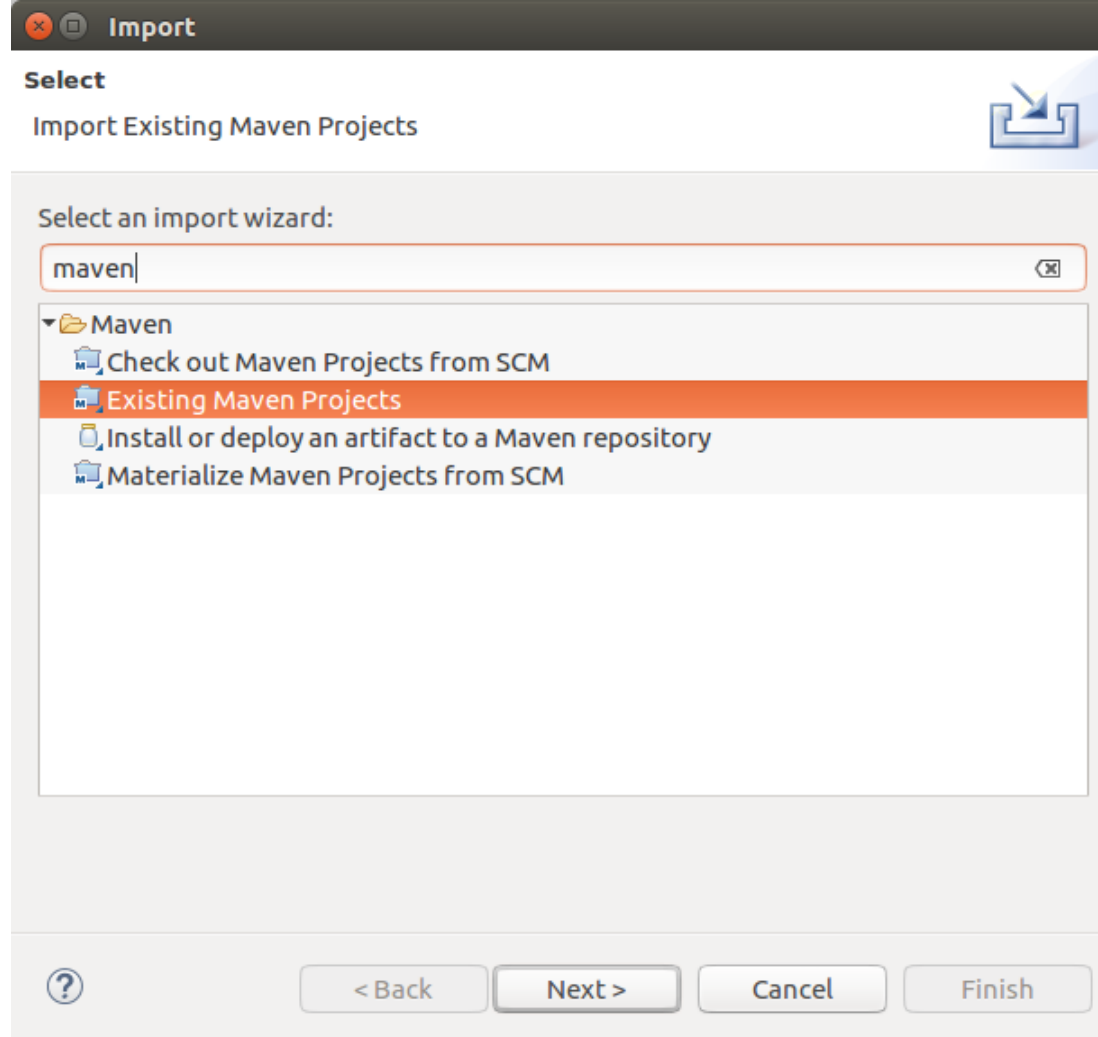




Clique com o botão direito na área branca à esquerda do Eclipse. Em seguida, escolha a opção `Import > Import...`. Você também pode realizar esse mesmo passo usando o menu `File > Import...`.



Na próxima tela, escolha a opção "Existing Maven Project" - você pode pesquisar pela palavra "maven" na caixa de busca, assim será mais fácil encontrar a opção.



Na próxima tela, clique em "Browse..." para selecionar o diretório onde o projeto foi descompactado. Selecione o diretório livraria .

Perceba que ao selecionar uma pasta que possui o projeto Maven, o Eclipse lista o arquivo `pom.xml` em "Projects". Para finalizar clique em "Finish".

Import Maven Projects

Maven Projects

Select Maven projects

Root Directory:

Browse...

Projects:

Select All

Deselect All

Select Tree

Deselect Tree

Refresh

☐ Add project(s) to working set

Advanced

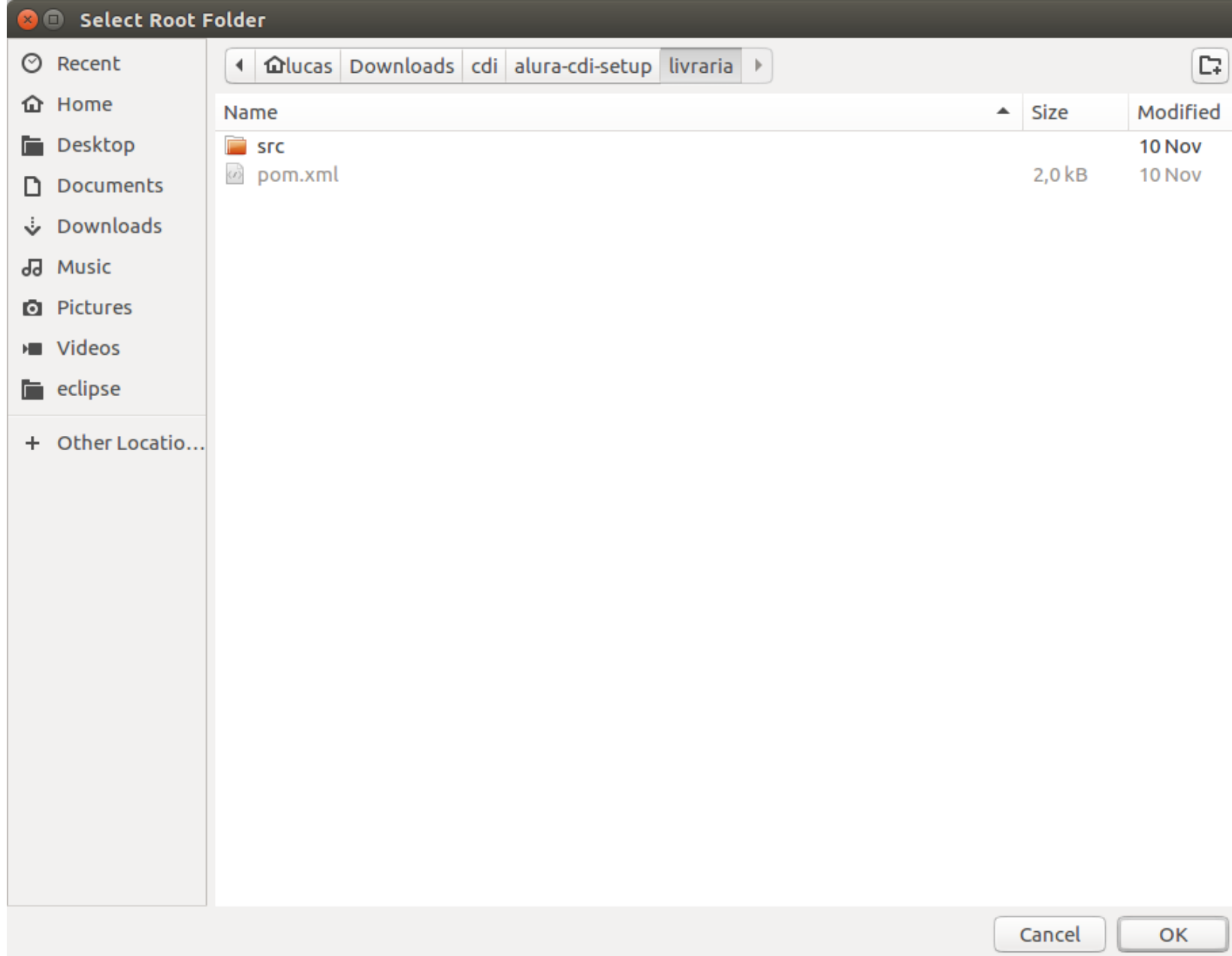
?

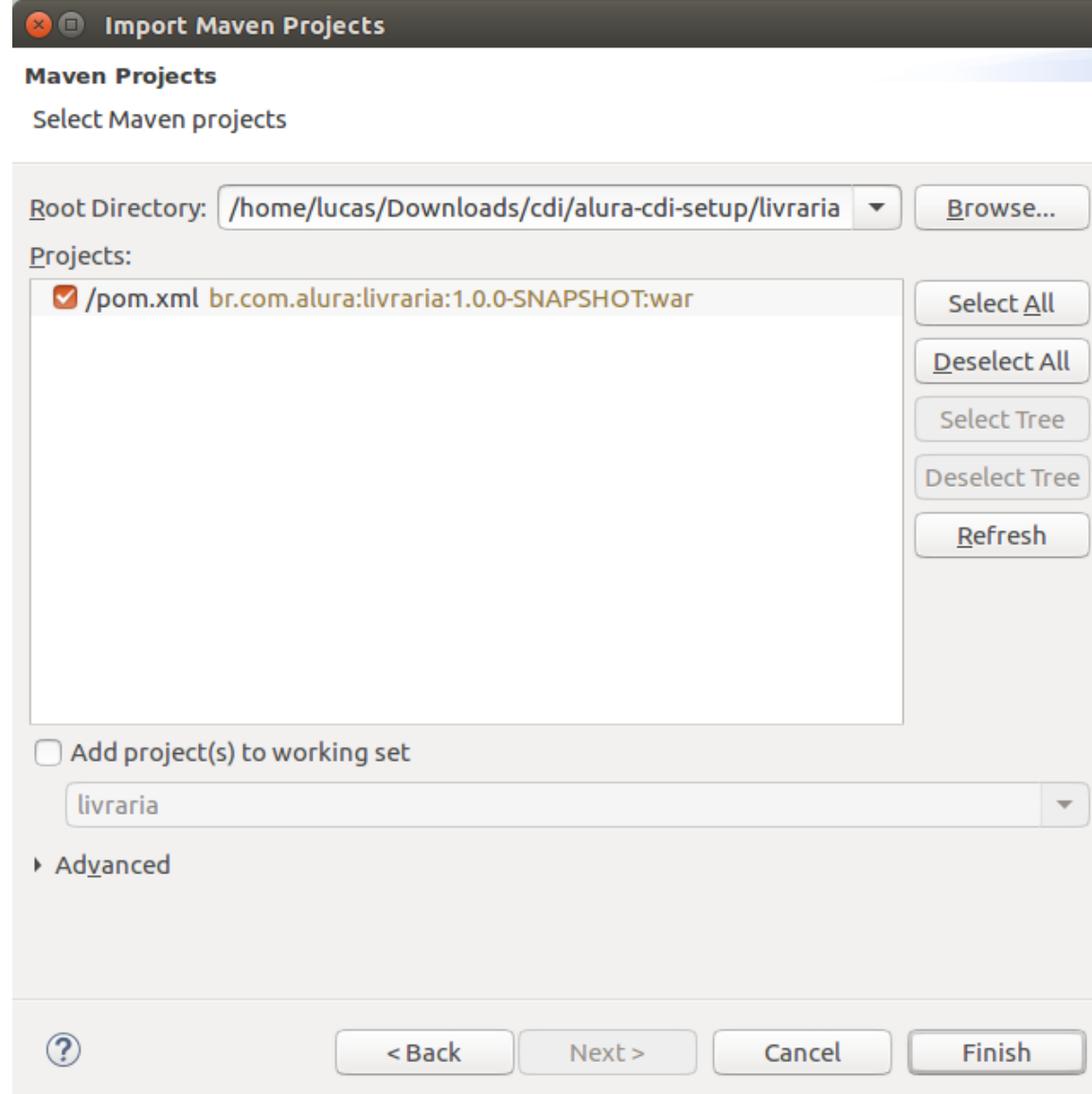
< Back

Next >

Cancel

Finish

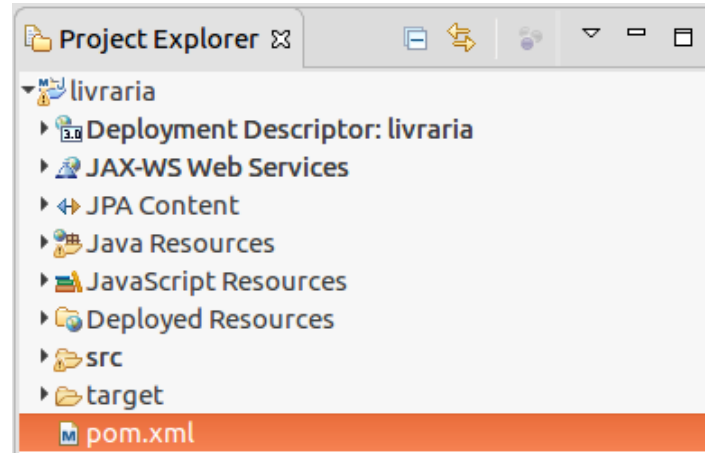




A primeira vez que você importar o projeto, talvez tenha que esperar um pouco, pois o Maven baixará todas as dependências do repositório central. Depois que ele fizer isto pela primeira vez, é feito um "cache" local, que sempre reutilizará as dependências locais que já foram baixadas.

Entendendo as dependências do projeto

Após a finalização da importação do projeto, vamos dar uma olhada nas dependências que o nosso projeto possui, abrindo o arquivo `pom.xml` .



No seguinte trecho de código, é possível ver que estamos utilizando a versão 8 do Java.

```
<properties>
  <maven.compiler.source>1.8</maven.compiler.source>
  <maven.compiler.target>1.8</maven.compiler.target>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>
```

COPIAR CÓDIGO

Na seguinte linha, que é possível encontrar logo no início do arquivo, indica que será gerado um pacote `.war` .

```
<packaging>war</packaging>
```

COPIAR CÓDIGO

O projeto utiliza o Hibernate. Em seguida temos duas dependências que são referentes ao JSF - já que este é utilizado pela aplicação.

```
<dependency>  
  <groupId>org.hibernate</groupId>  
  <artifactId>hibernate-entitymanager</artifactId>  
  <version>4.1.8.Final</version>  
</dependency>
```

```
<dependency>  
  <groupId>com.sun.faces</groupId>  
  <artifactId>jsf-api</artifactId>  
  <version>2.2.13</version>  
</dependency>
```

```
<dependency>  
  <groupId>com.sun.faces</groupId>  
  <artifactId>jsf-impl</artifactId>  
  <version>2.2.13</version>  
</dependency>
```

[COPIAR CÓDIGO](#)

A próxima dependência é o driver do MySQL, para que seja possível se conectar com o banco de dados.

```
<dependency>  
  <groupId>mysql</groupId>  
  <artifactId>mysql-connector-java</artifactId>  
  <version>5.1.22</version>  
</dependency>
```

[COPIAR CÓDIGO](#)

A dependência do `hibernate-validator` está inclusa porque o projeto utiliza algumas coisas da Bean Validation.

```
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-validator</artifactId>
  <version>4.3.0.Final</version>
</dependency>
```

COPIAR CÓDIGO

E por fim, duas dependências referentes ao [PrimeFaces](http://www.primefaces.org/) (<http://www.primefaces.org/>): O PrimeFaces em si e seus temas.

```
<dependency>
  <groupId>org.primefaces.themes</groupId>
  <artifactId>all-themes</artifactId>
  <version>1.0.10</version>
</dependency>
```

```
<dependency>
  <groupId>org.primefaces</groupId>
  <artifactId>primefaces</artifactId>
  <version>5.3</version>
</dependency>
```

COPIAR CÓDIGO

O PrimeFaces pede que seja adicionado seu repositório para obter a dependência do tema. Portanto, no seguinte trecho de código, temos adicionada a configuração do repositório.

```
<repositories>
  <repository>
    <id>prime-repo</id>
    <name>PrimeFaces Maven Repository</name>
```

```
        <url>http://repository.primefaces.org</url>
        <layout>default</layout>
    </repository>
</repositories>
```

[COPIAR CÓDIGO](#)

Para finalizar, último a *tag* `finalName` define o nome do artefato gerado, que nesse caso será `livraria.war` . E foi também adicionado um *plugin* do Maven para gerar os nossos `wars` .

```
<build>
    <finalName>livraria</finalName>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-war-plugin</artifactId>
            <version>2.6</version>
        </plugin>
    </plugins>
</build>
```

[COPIAR CÓDIGO](#)

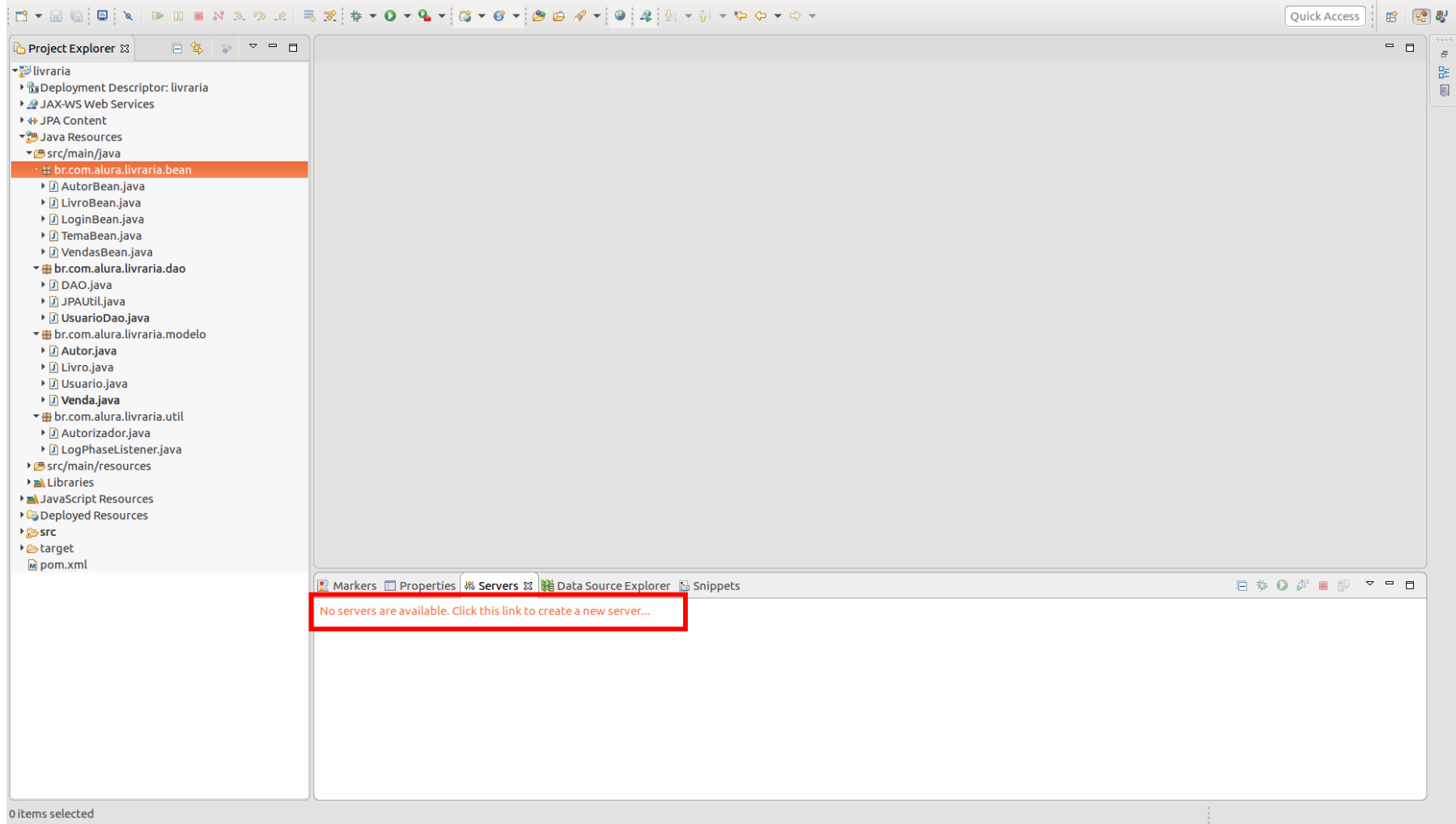
Ao expandir a pasta `Java Resources > src/main/java` , podemos ver os pacotes e classes da aplicação. Os `beans` estão relacionados com nossas *views*, os arquivos `.xhtml` . Temos também os DAOs, os modelos e classes utilitárias.



Configurando o Servlet Container

Agora que conseguimos importar nossa aplicação, vamos precisar de alguém para rodar a aplicação Web. Nesse momento não vamos utilizar um servidor de aplicação, mas um Servlet Container, o [Tomcat \(http://tomcat.apache.org/\)](http://tomcat.apache.org/). Após fazer o Download do Tomcat, descompacte em um diretório de sua preferência.

No Eclipse, na aba Servers, clique na mensagem "No servers are available. Click this link to create a new server...".



Na tela que irá parecer, Selecione Apache > Tomcat v8.0 Server e clique em "Next".

New Server

Define a New Server

Choose the type of server to create

Select the server type:

type filter text

- Tomcat v5.0 Server
- Tomcat v5.5 Server
- Tomcat v6.0 Server
- Tomcat v7.0 Server
- Tomcat v8.0 Server**
- Tomcat v8.5 Server
- Tomcat v9.0 Server
- Basic
- Cloud Foundry

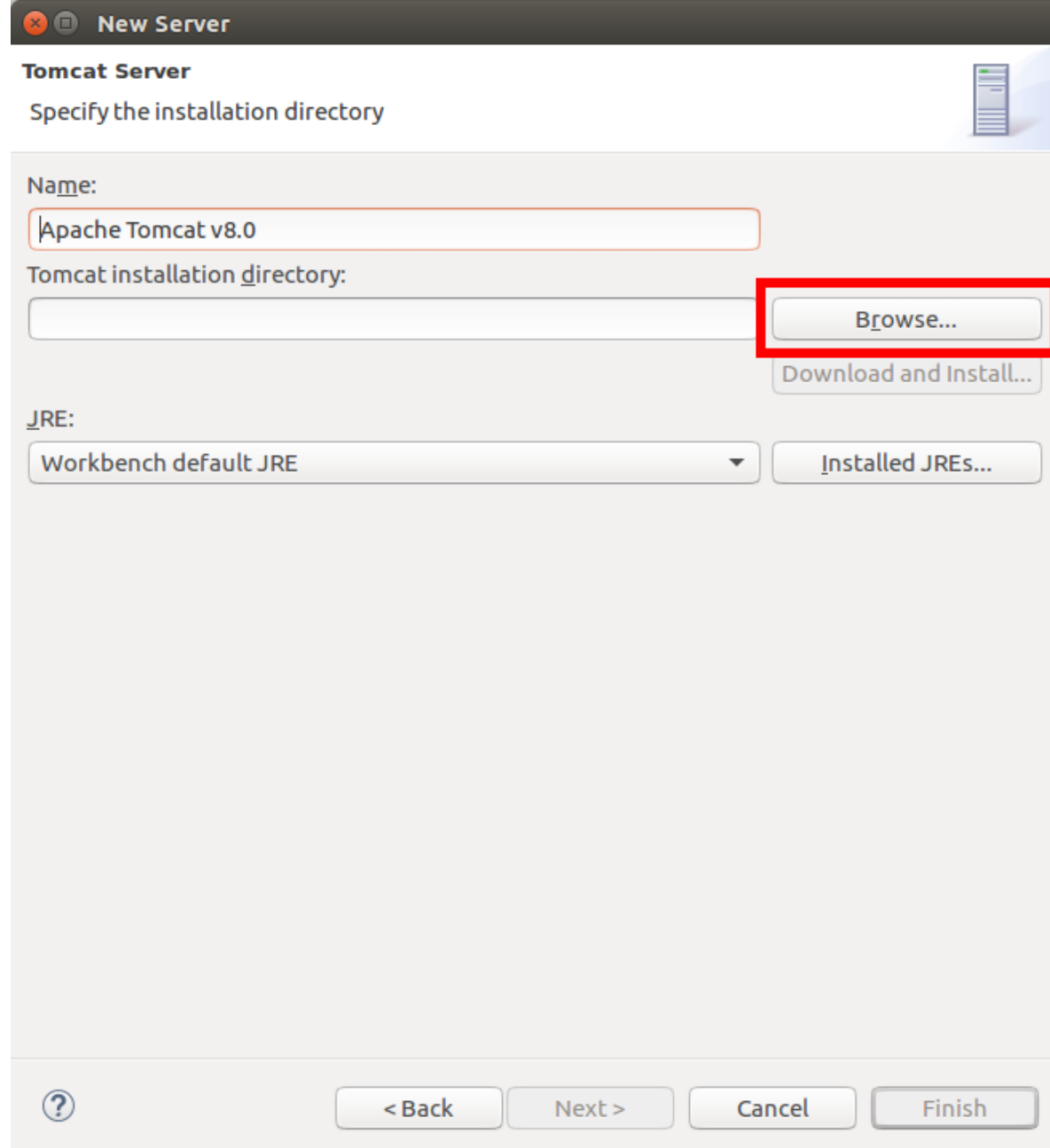
Publishes and runs J2EE and Java EE Web projects and server configurations to a local Tomcat server.

Server's host name: localhost

Server name: Tomcat v8.0 Server at localhost

? < Back Next > Cancel Finish

Na próxima tela, clique em "Browser" e selecione o diretório do Tomcat. Feito isso, clique em "Finish".

The image shows a 'New Server' dialog box in an IDE. It has a title bar with a close button and the text 'New Server'. The main area is titled 'Tomcat Server' with a subtitle 'Specify the installation directory' and a server icon. There are three main sections: 'Name:' with a text field containing 'Apache Tomcat v8.0'; 'Tomcat installation directory:' with an empty text field and a 'Browse...' button highlighted by a red rectangle; and 'JRE:' with a dropdown menu showing 'Workbench default JRE' and an 'Installed JREs...' button. At the bottom, there is a question mark icon and four buttons: '< Back', 'Next >', 'Cancel', and 'Finish'.

New Server

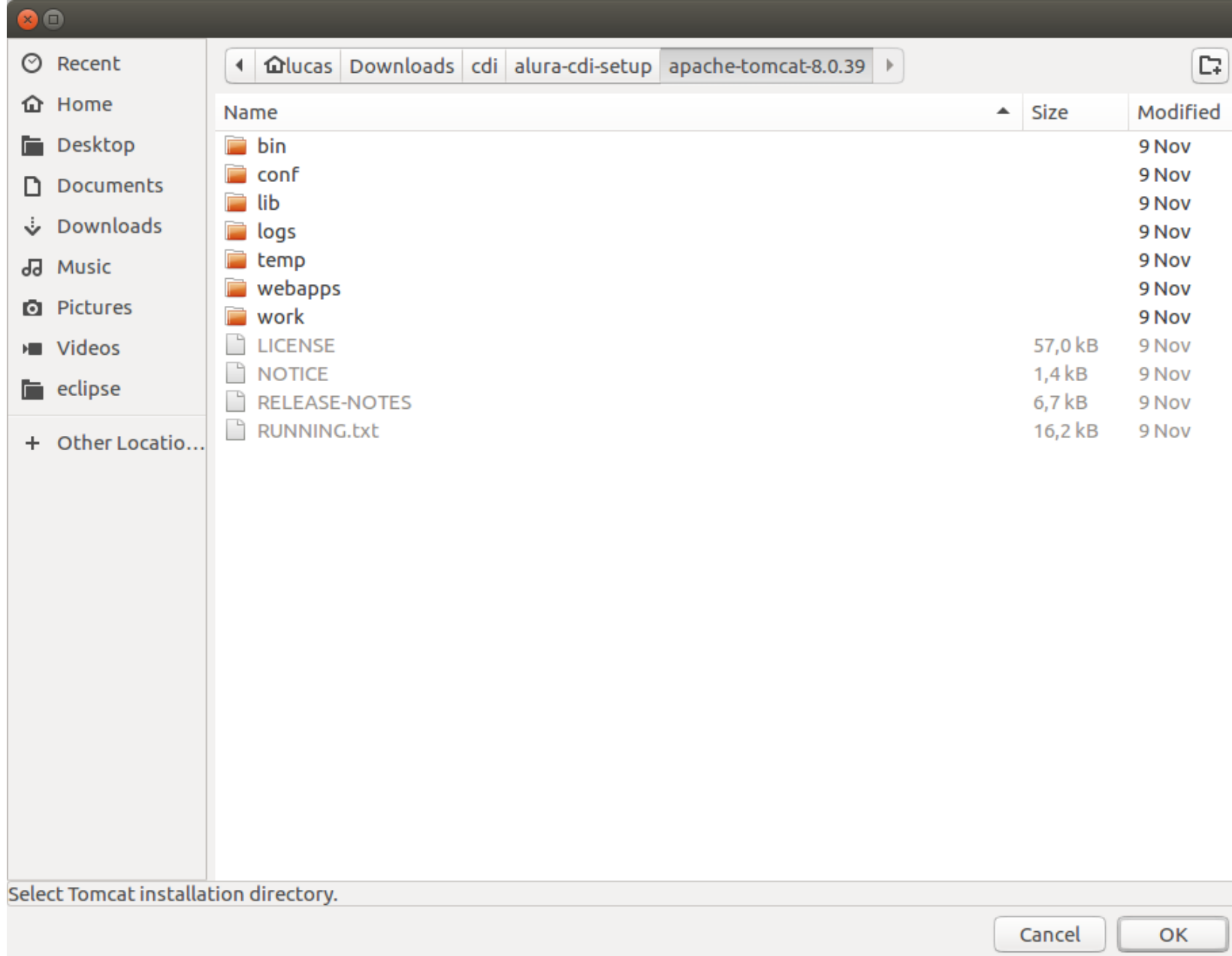
Tomcat Server
Specify the installation directory

Name:
Apache Tomcat v8.0

Tomcat installation directory:
 Browse...
Download and Install...

JRE:
Workbench default JRE

? < Back Next > Cancel Finish



New Server

Tomcat Server

Specify the installation directory

Name:
Apache Tomcat v8.0

Tomcat installation directory:
/home/lucas/Downloads/cdi/alura-cdi-setup/apache-tomcat-8.0.3

Browse...

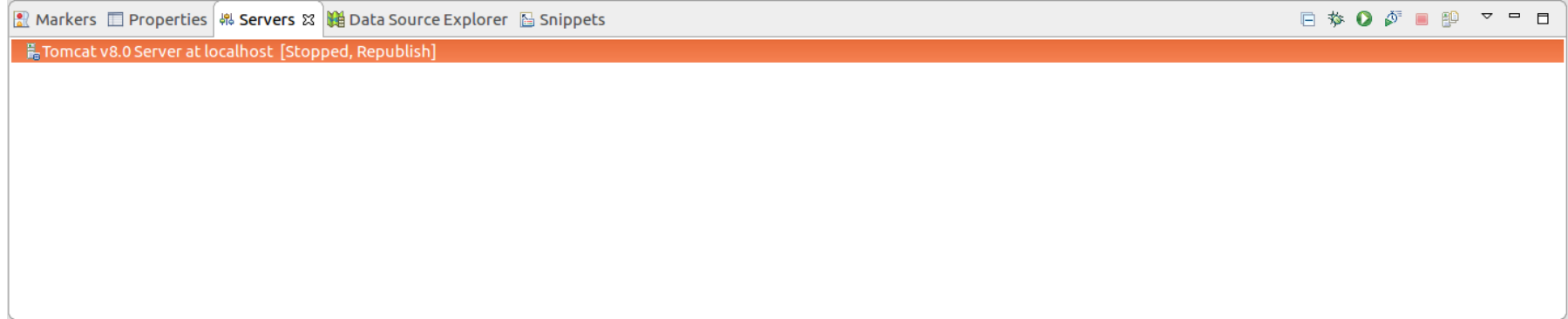
Download and Install...

JRE:
Workbench default JRE

Installed JREs...

? < Back Next > Cancel Finish

Na aba Servers, já é possível ver o Tomcat configurado.



Configurando o banco de dados

Resta um último passo. Nossa aplicação se comunicará com o banco de dados. Se você olhar dentro do diretório `src/main/resources > META-INF`, existe o arquivo `persistence.xml`. O `persistence.xml` é o arquivo de configuração para que a aplicação possa se comunicar com o banco de dados.

Devido a seguinte linha do `persistence.xml`, nossa aplicação espera que exista um banco de dados chamado `livrariadb`. Vamos criar esse banco de dados agora.

```
<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost/livrariadb" />
```

[COPIAR CÓDIGO](#)

Segundo nossa configuração, o banco de dados MySQL não possui senha:

```
<property name="javax.persistence.jdbc.password" value="" />
```

[COPIAR CÓDIGO](#)

Vamos nos conectar ao servidor do MySQL:

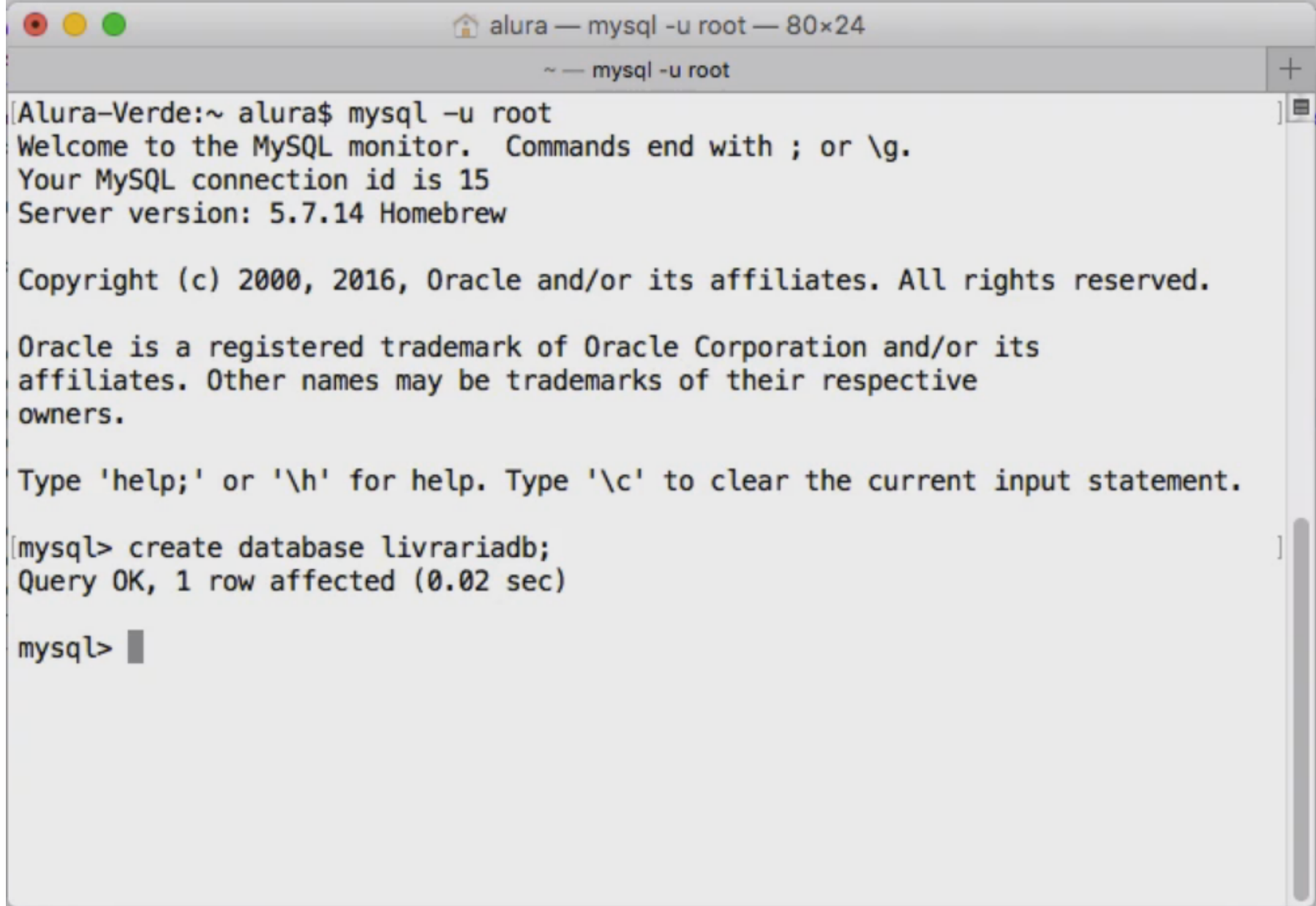
```
$ mysql -u root
```

[COPIAR CÓDIGO](#)

E em seguida, já conectados, vamos criar o banco de dados:

```
> create database livrariadb;
```

[COPIAR CÓDIGO](#)

A screenshot of a macOS terminal window titled 'alura — mysql -u root — 80x24'. The terminal shows the execution of 'mysql -u root' from the prompt 'Alura-Verde:~ alura\$'. The MySQL monitor displays a welcome message, connection ID 15, and server version 5.7.14 Homebrew. It then shows the creation of a database named 'livrariadb' with the command 'create database livrariadb;', which returns 'Query OK, 1 row affected (0.02 sec)'. The prompt 'mysql>' is shown with a cursor. The terminal window has standard macOS window controls (red, yellow, green buttons) and a scrollbar on the right.

```
alura — mysql -u root — 80x24
~ — mysql -u root

[Alura-Verde:~ alura$ mysql -u root
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 15
Server version: 5.7.14 Homebrew

Copyright (c) 2000, 2016, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

[mysql> create database livrariadb;
Query OK, 1 row affected (0.02 sec)

mysql> █
```

Ao acessarmos o banco, e listarmos as tabelas, vamos ver que não existe nenhuma tabela porque acabamos de criar o banco.

```
> use livrariadb;
Database changed
```

```
> show tables;  
Empty set (0,00 sec)
```

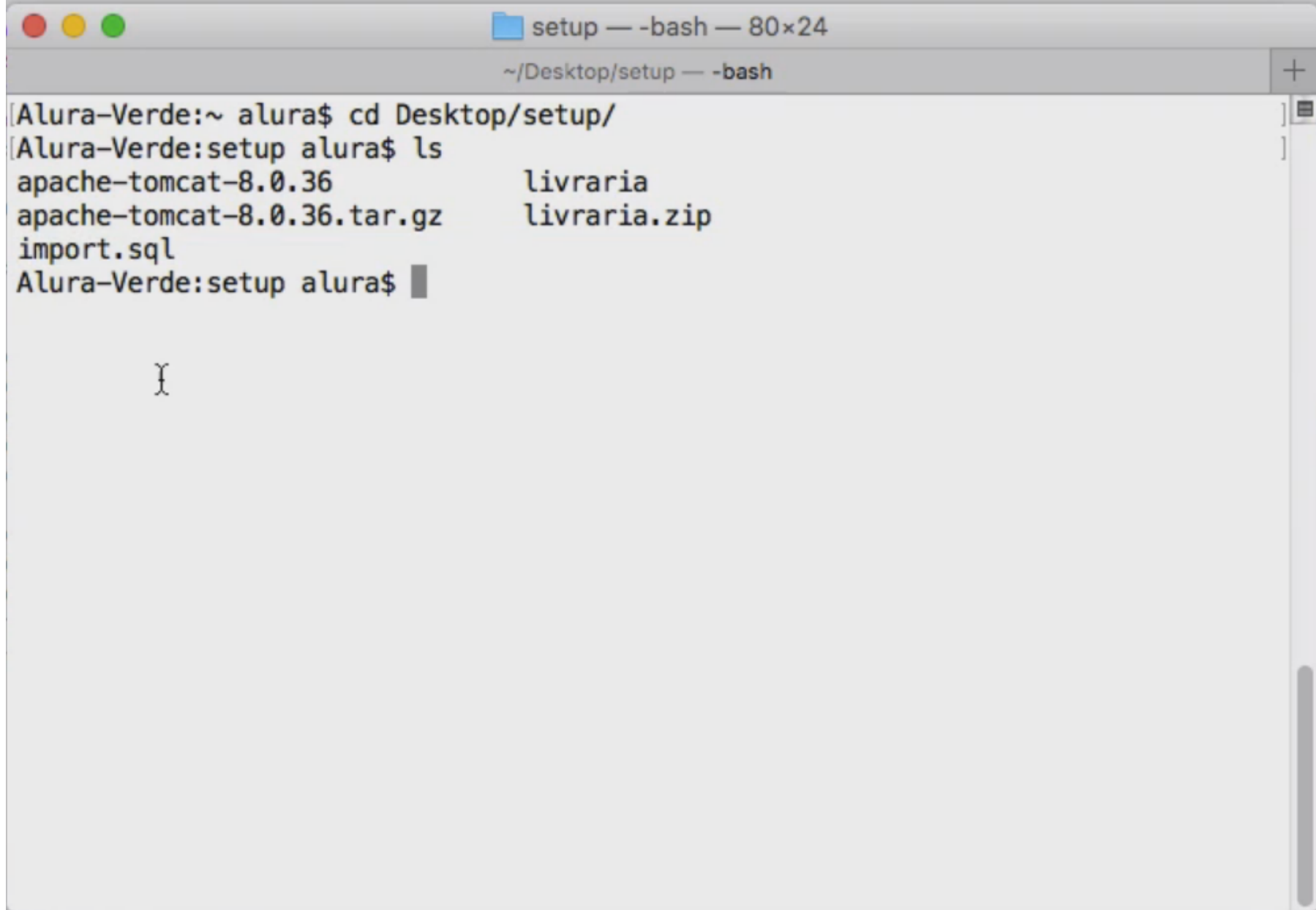
[COPIAR CÓDIGO](#)

Para iniciarmos com alguns dados e não seja preciso que esses dados tenham que ser cadastrados manualmente, vamos importar um *script*. Saia do `mysql` utilizando o comando **exit**.

```
> exit
```

[COPIAR CÓDIGO](#)

Vamos utilizar o comando `cd` e navegar até o diretório onde se encontra o arquivo `import.sql`. Dentro desse arquivo existem vários comandos SQL que desejamos que o MySQL execute.

A terminal window titled 'setup — -bash — 80x24' with a subtitle '~ / Desktop / setup — -bash'. The window shows a user named 'alura' navigating to the 'Desktop/setup/' directory and listing its contents. The output shows four files: 'apache-tomcat-8.0.36', 'apache-tomcat-8.0.36.tar.gz', 'import.sql', and 'livraria.zip'. The prompt 'Alura-Verde:setup alura\$' is shown at the bottom with a cursor.

```
[Alura-Verde:~ alura$ cd Desktop/setup/
[Alura-Verde:setup alura$ ls
apache-tomcat-8.0.36      livraria
apache-tomcat-8.0.36.tar.gz livraria.zip
import.sql
Alura-Verde:setup alura$
```

Podemos ver o conteúdo do arquivo utilizando o comando `cat` . O arquivo possui comandos para excluir as tabelas caso existam e em seguida criá-las. Também são inseridos alguns dados no banco. Você pode verificar o conteúdo completo na sua máquina ao executar o comando `cat` , ou abrindo o arquivo no seu editor de textos preferido.

```
$ cat import.sql
```

```
DROP TABLE IF EXISTS `Autor`;
/*!40101 SET @saved_cs_client      = @@character_set_client */;
/*!40101 SET character_set_client = utf8 */;
CREATE TABLE `Autor` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `email` varchar(255) DEFAULT NULL,
  `nome` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=14 DEFAULT CHARSET=utf8;
/*!40101 SET character_set_client = @saved_cs_client */;

--
-- Dumping data for table `Autor`
--

LOCK TABLES `Autor` WRITE;
/*!40000 ALTER TABLE `Autor` DISABLE KEYS */;
INSERT INTO `Autor` VALUES (1,'sergio.lopes@caelum.com.br','Sergio Lopes'),(2,'nico.steppat@caelum.
/*!40000 ALTER TABLE `Autor` ENABLE KEYS */;
UNLOCK TABLES;

/* restante do arquivo... */
```

COPIAR CÓDIGO

Queremos que o *script* seja executado dentro do banco de dados livraria. Para importar, utilizamos o comando `mysql` , fornecendo os dados de login, o nome do banco no qual desejamos executar o *script* e o caminho para o *script*.


```
$ mysql -u root livrariadb < import.sql
```

[COPIAR CÓDIGO](#)

Após digitar o comando, pressione "Enter". Pode demorar alguns segundos. Vamos logar novamente no MySQL para conferir se os dados foram importados.

Vamos utilizar o comando `use livrariadb` para selecionar o banco de dados e em seguida o comando `show tables` para listar as tabelas.

```
$ mysql -u root livrariadb
```

```
> use livrariadb;
```

```
> show tables;
```

```
+-----+
| Tables_in_livrariadb |
+-----+
| Autor                |
| Livro                |
| Livro_Autor          |
| Usuario              |
+-----+
4 rows in set (0,00 sec)
```

[COPIAR CÓDIGO](#)

Agora vamos fazer um *querie* para verificar se foi inserido o usuário `admin` com a senha `12345` na tabela `Usuario` .

```
> select * from Usuario;
+----+-----+-----+
| id | email                | senha |
+----+-----+-----+
|  1 | admin@caelum.com.br | 12345 |
+----+-----+-----+
1 row in set (0,00 sec)
```

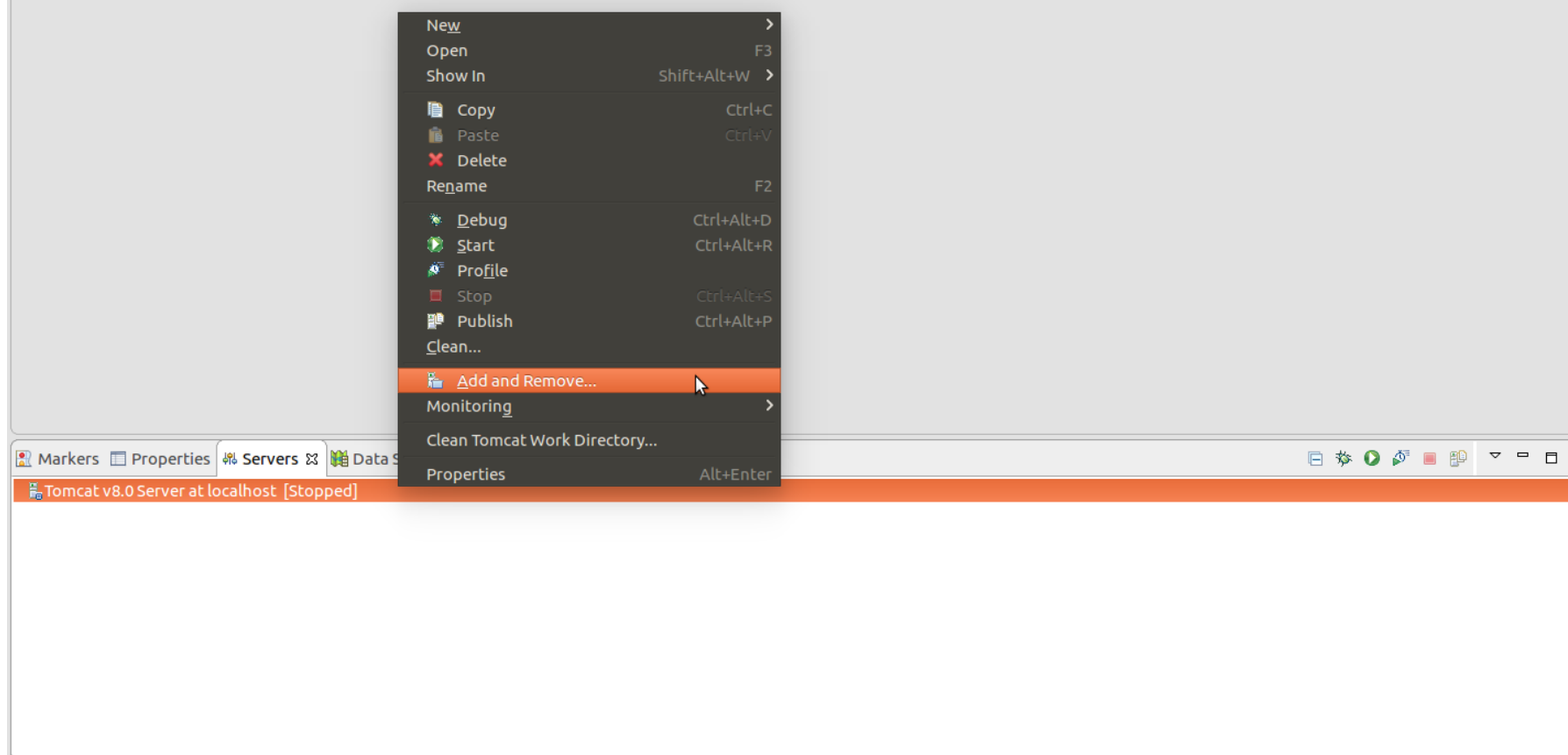
[COPIAR CÓDIGO](#)

Pronto! Podemos ver que um registro já foi inserido. Temos nosso banco de dados totalmente populado. Desta forma, não precisamos ficar nos preocupando em cadastrar as informações manualmente.

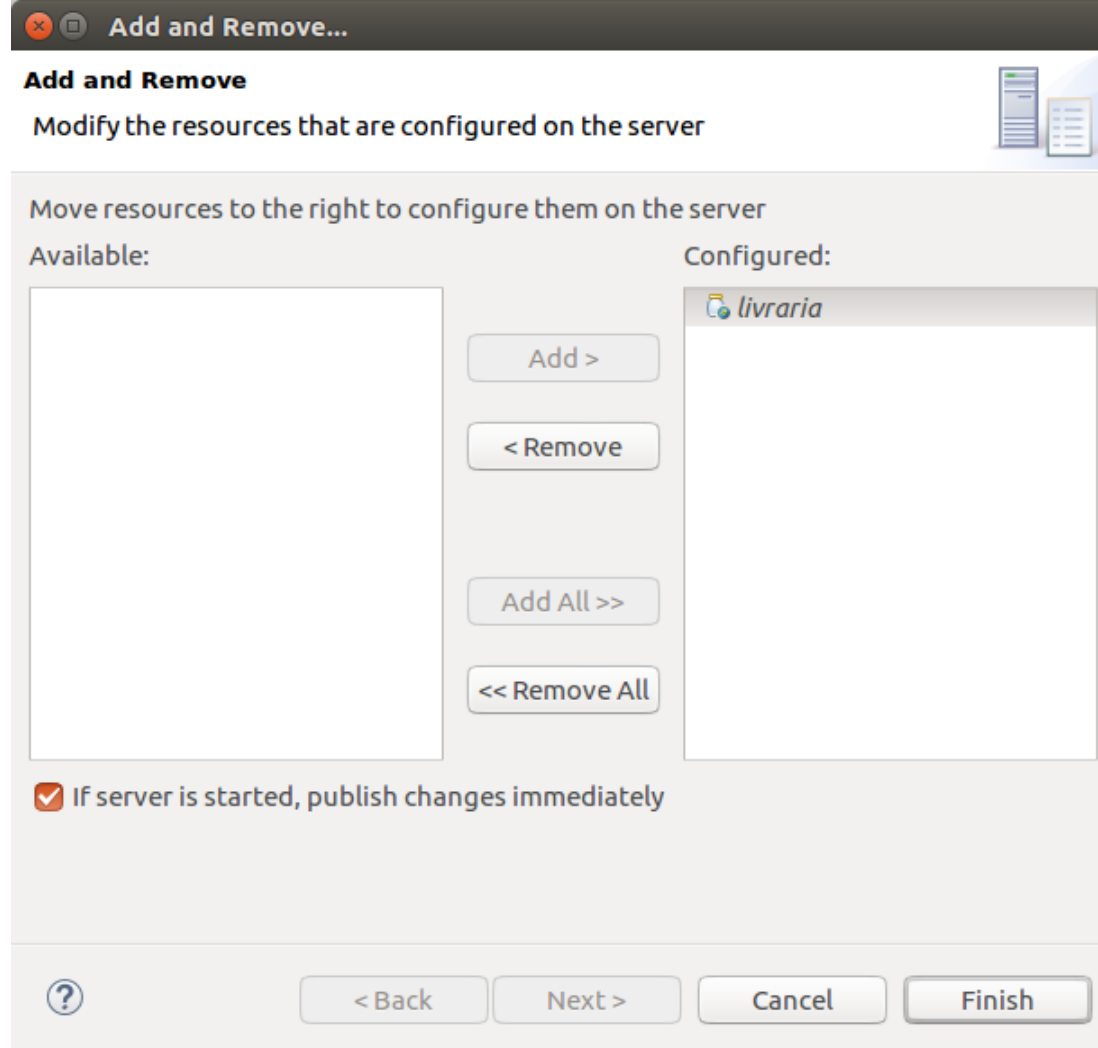
Fazendo deploy do projeto

Agora que configuramos o Tomcat e banco de dados, vamos subir nossa aplicação.

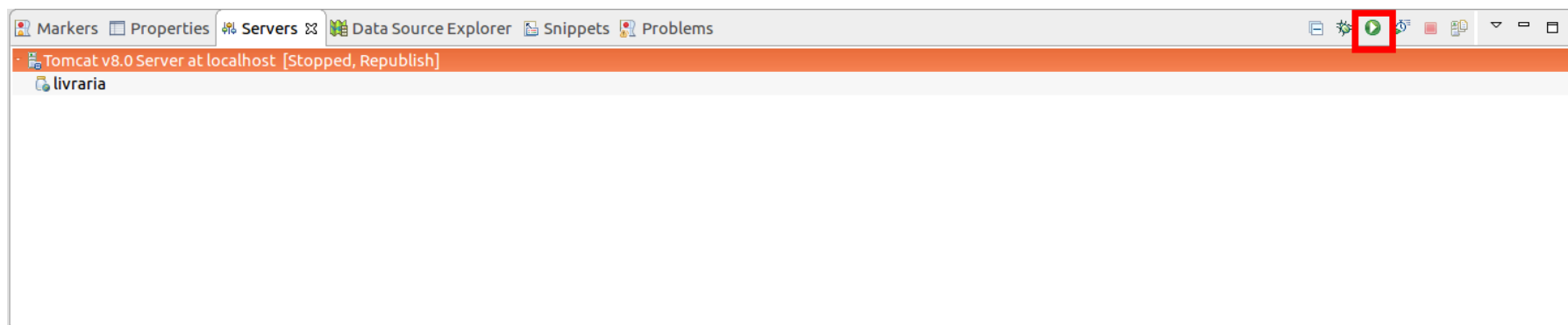
No Eclipse, na aba Servers, clique com o botão direito no Tomcat e escolha a opção "Add and Remove".



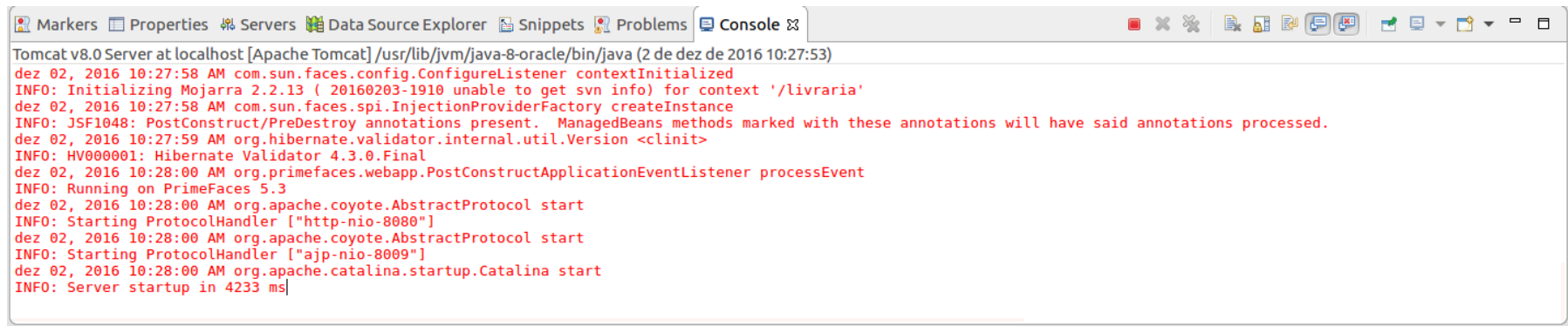
Na tela que irá aparecer, selecione o projeto e clique no botão "Add". Em seguida, clique em "Finish".



Clique no botão destacado na imagem seguinte, para iniciar o Tomcat.



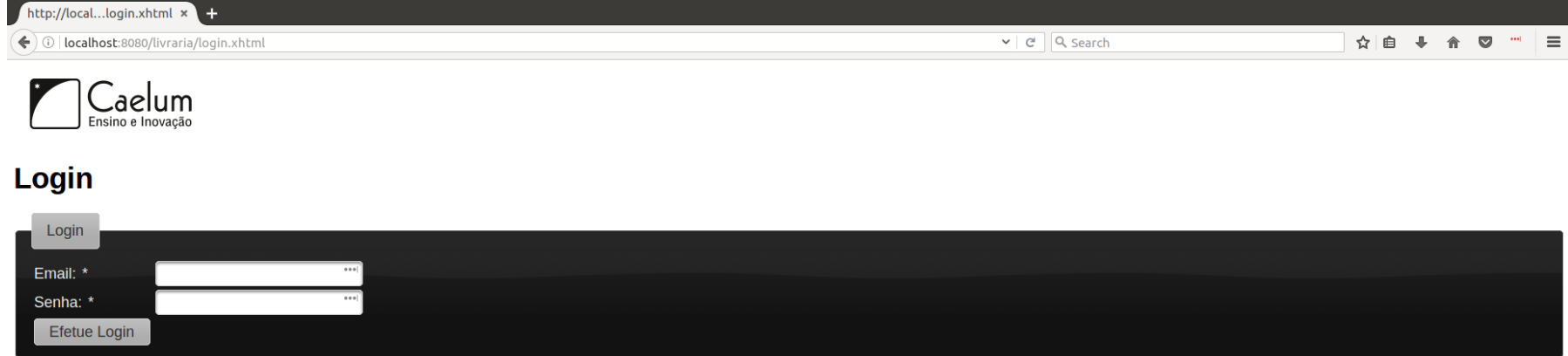
Não recebemos nenhuma mensagem de erro no Console, então aparentemente, tudo ok.



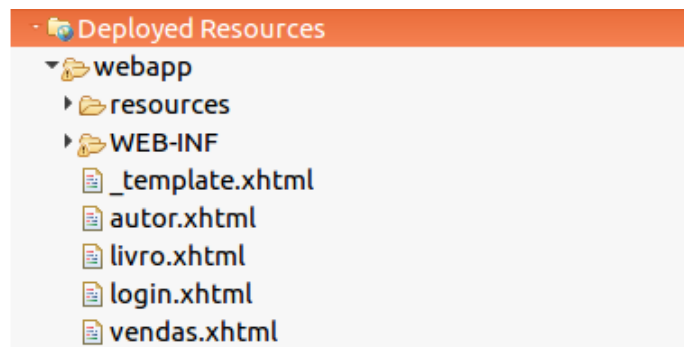
The screenshot shows the Eclipse IDE's Console window. The title bar indicates the console is for 'Tomcat v8.0 Server at localhost [Apache Tomcat]' and shows the path '/usr/lib/jvm/java-8-oracle/bin/java' and the time '2 de dez de 2016 10:27:53'. The console output displays the following log messages:

```
dez 02, 2016 10:27:58 AM com.sun.faces.config.ConfigureListener contextInitialized
INFO: Initializing Mojarra 2.2.13 ( 20160203-1910 unable to get svn info) for context '/livraria'
dez 02, 2016 10:27:58 AM com.sun.faces.spi.InjectionProviderFactory createInstance
INFO: JSF1048: PostConstruct/PreDestroy annotations present. ManagedBeans methods marked with these annotations will have said annotations processed.
dez 02, 2016 10:27:59 AM org.hibernate.validator.internal.util.Version <clinit>
INFO: HV000001: Hibernate Validator 4.3.0.Final
dez 02, 2016 10:28:00 AM org.primefaces.webapp.PostConstructApplicationEventListener processEvent
INFO: Running on PrimeFaces 5.3
dez 02, 2016 10:28:00 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8080"]
dez 02, 2016 10:28:00 AM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
dez 02, 2016 10:28:00 AM org.apache.catalina.startup.Catalina start
INFO: Server startup in 4233 ms
```

Para conferir a aplicação, basta acessar <http://localhost:8080/livraria/login.xhtml> (<http://localhost:8080/livraria/login.xhtml>) no seu navegador.



Para verificar de onde vem o Login, se você expandir a pasta Deployed Resources > webapp , poderá ver que lá se encontram os arquivos que podemos acessar.




Se você tentar acessar [a página dos livros \(http://localhost:8080/livraria/livro.xhtml\)](http://localhost:8080/livraria/livro.xhtml), será direcionado para a página de login. Existe uma regra na classe `Autorizador` (pacote `br.com.alura.livraria.util`) que indica quando você não possui um usuário logado, então, a aplicação redirecionará para a página de login.

Você pode efetuar o login utilizando os dados que buscamos anteriormente no banco:

usuário: `admin@caelum.com.br` , senha: `12345` .

Após o login temos acesso à página de livros.

localhost:8080/livro.html



vader

Menu

Novo Livro

Dados do Livro

Título: *

ISBN:

Preço:

Data de Lançamento:

0.0

02/12/2016

Dados do Autor

Selecione Autor:

Sergio Lopes

Gravar Autor

ou cadastrar novo autor

Nenhum autor

Gravar Livro

Livros					
1 2					
Título	ISBN	Preço	Data	Alterar	Remover
MEAN	121-3-12-312312-3	R\$ 49,90	26/02/2016	alterar	remover
Arquitetura Java	123-1-31-212313-1	R\$ 49,90	27/02/2016	alterar	remover
AngularJS	123-1-31-231312-3	R\$ 39,90	01/03/2016	alterar	remover
TDD	124-5-55-533223-2	R\$ 39,90	01/03/2016	alterar	remover

O intuito desta aula era preparar todo o ambiente. Agora que está tudo configurado e funcionando, mais adiante começaremos a utilizar o CDI. Até lá!