**Atividade Prática Prévia**

|  |  |
| --- | --- |
| **Disciplina** | **ARI – Arquitetura de Integração** |
| **Aluno (a)** | **Sidney Hatada** |

**Objetivos e orientações gerais sobre este trabalho prévio:**

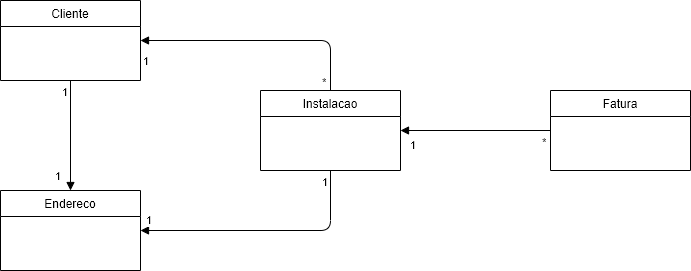
* Este trabalho deverá ser feito individualmente;
* O trabalho tem o objetivo de aproximar o(a) aluno(a) de um exemplo prático de uma solução simplificada para integração baseada em serviços REST;
* Deverá ser entregue como resultado da atividade um documento contendo a análise crítica da arquitetura da solução, com um link para o repositório GitHub do(a) aluno(a) que contenha os códigos fontes da solução relativa à execução dos passos desta atividade.

**Cenário de Estudo**

Uma empresa do ramo de energia elétrica denominada Acme Producer (AP) tem um processo de negócio em comum onde é necessária uma integração entre sistemas.

O sistema da empresa AP produz as seguintes informações:

* Dados de cliente: CPF, nome, data de nascimento, endereço de cobrança;
* Dados de instalação: código da instalação, endereço da instalação;
* Dados de Fatura: código da fatura, código da instalação, CPF do cliente, data de vencimento, data de leitura, valor da leitura, valor da conta;



Todas as informações acima são disponibilizadas através de uma API Rest pela empresa AP.

Os clientes da empresa AP precisam registrar as informações de suas contas de energia. Para tal, realizam uma integração com a empresa AP em que recuperam as informações consumindo a API e importam para seu banco de dados para auditoria posterior.

Os serviços oferecidos pela empresa AP estão listados abaixo:

Cliente:

* Consultar a lista de clientes cadastrados;
* Consultar cliente cadastrado pelo seu CPF;
* Cadastrar um cliente.

Instalações

* Consultar a lista de instalações cadastradas;
* Consultar uma instalação pelo seu código;
* Consultar instalações de um cliente pelo CPF;
* Cadastrar uma instalação.

Faturas:

* Consultar a lista de faturas geradas;
* Consultar fatura cadastrado pelo seu código;
* Consultar faturas de um cliente pelo CPF;
* Gerar uma fatura.

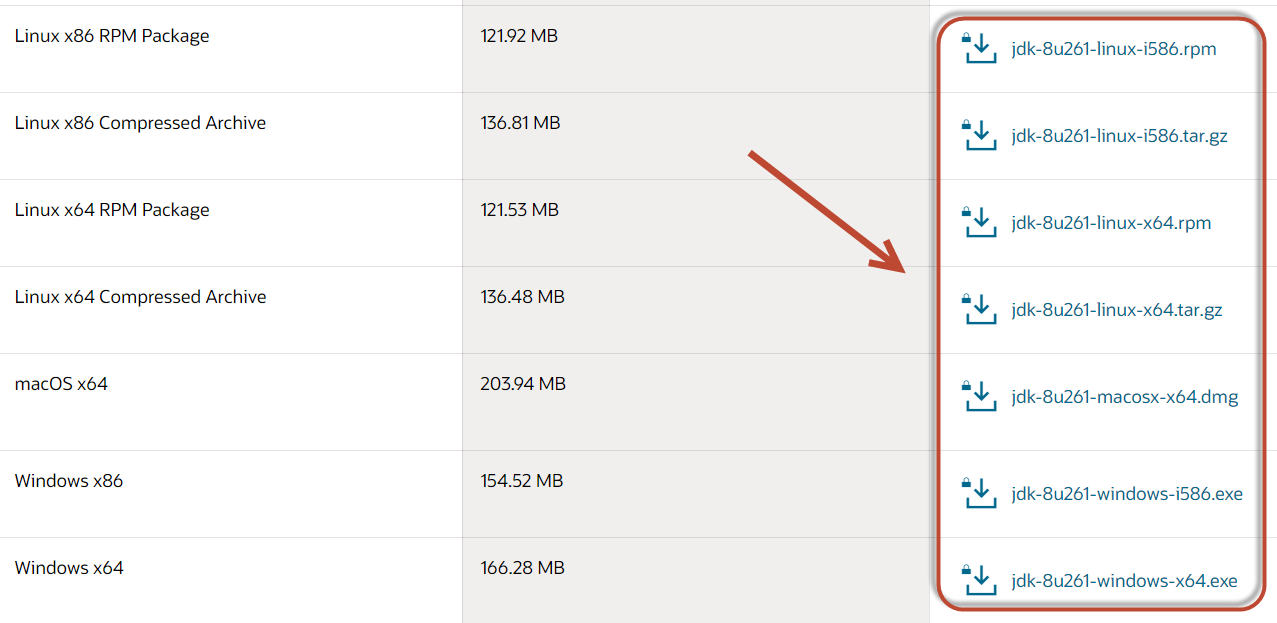
**Atividades**

O(A) aluno(a) deverá desempenhar as seguintes atividades:

1. Instalação e configuração de um ambiente de desenvolvimento conforme passos descritos abaixo. Se você já possui um ambiente de desenvolvimento, pode pular essa etapa e ir direto para o passo 2.

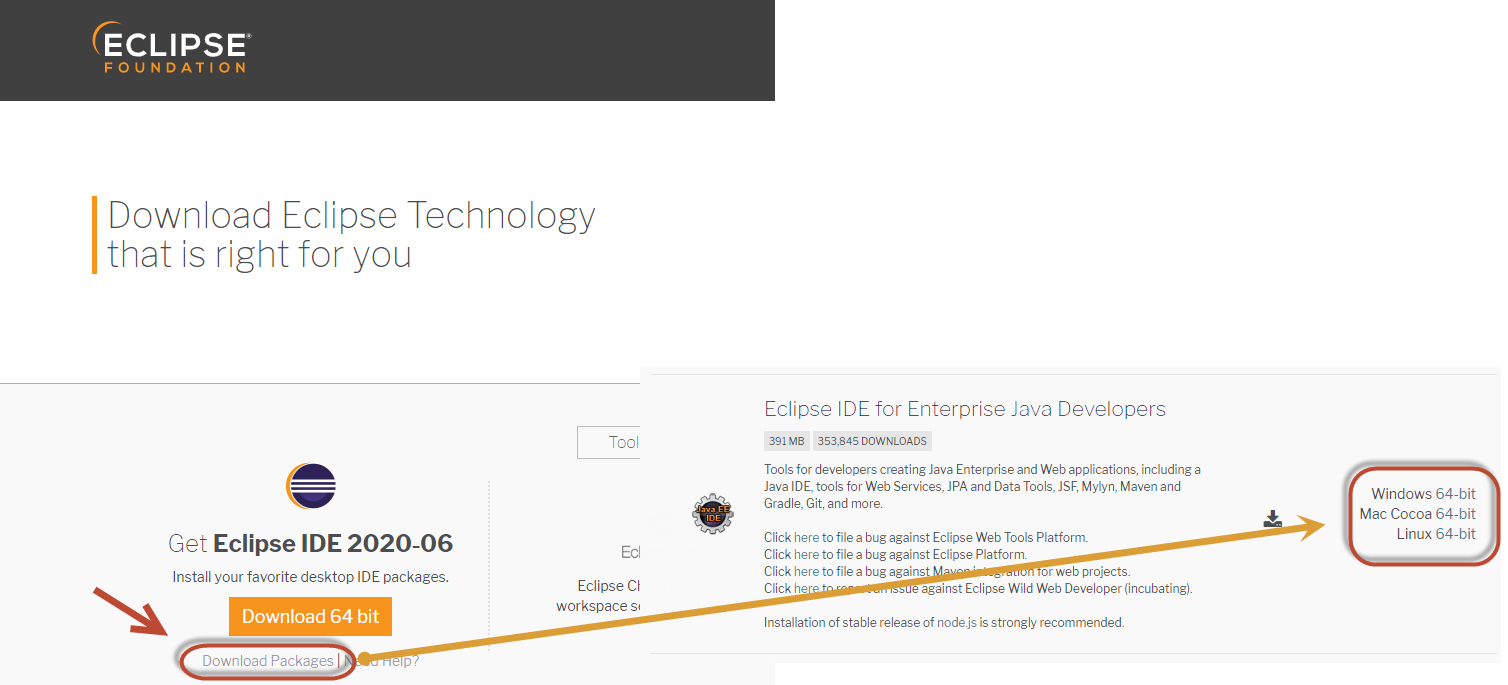
Download do JAVA Development Kit (JDK):

* Acessar o link: <https://www.oracle.com/br/java/technologies/javase/javase-jdk8-downloads.html>;
* Selecionar a opção de download de acordo com o seu sistema operacional;
* Instalar a o JDK.



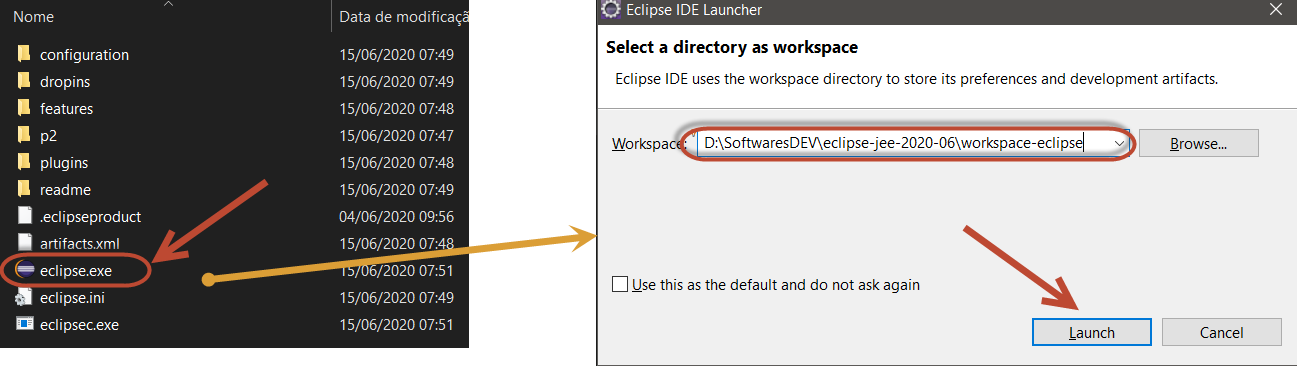
Download da IDE de desenvolvimento:

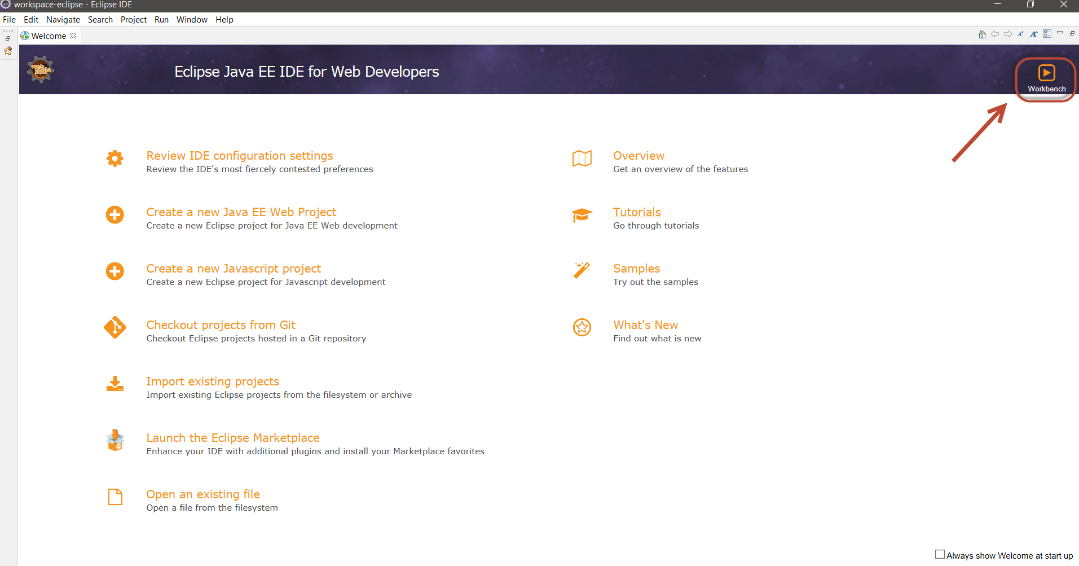
* Acessar o link: <https://www.eclipse.org/downloads/>;
* Selecionar a opção *Download Packages* e escolher a opção *Eclipse IDE for Enterprise Java Developers;*
* Realizar o download da versão compatível com o seu sistema operacional (SO).



Instalação da IDE de desenvolvimento:

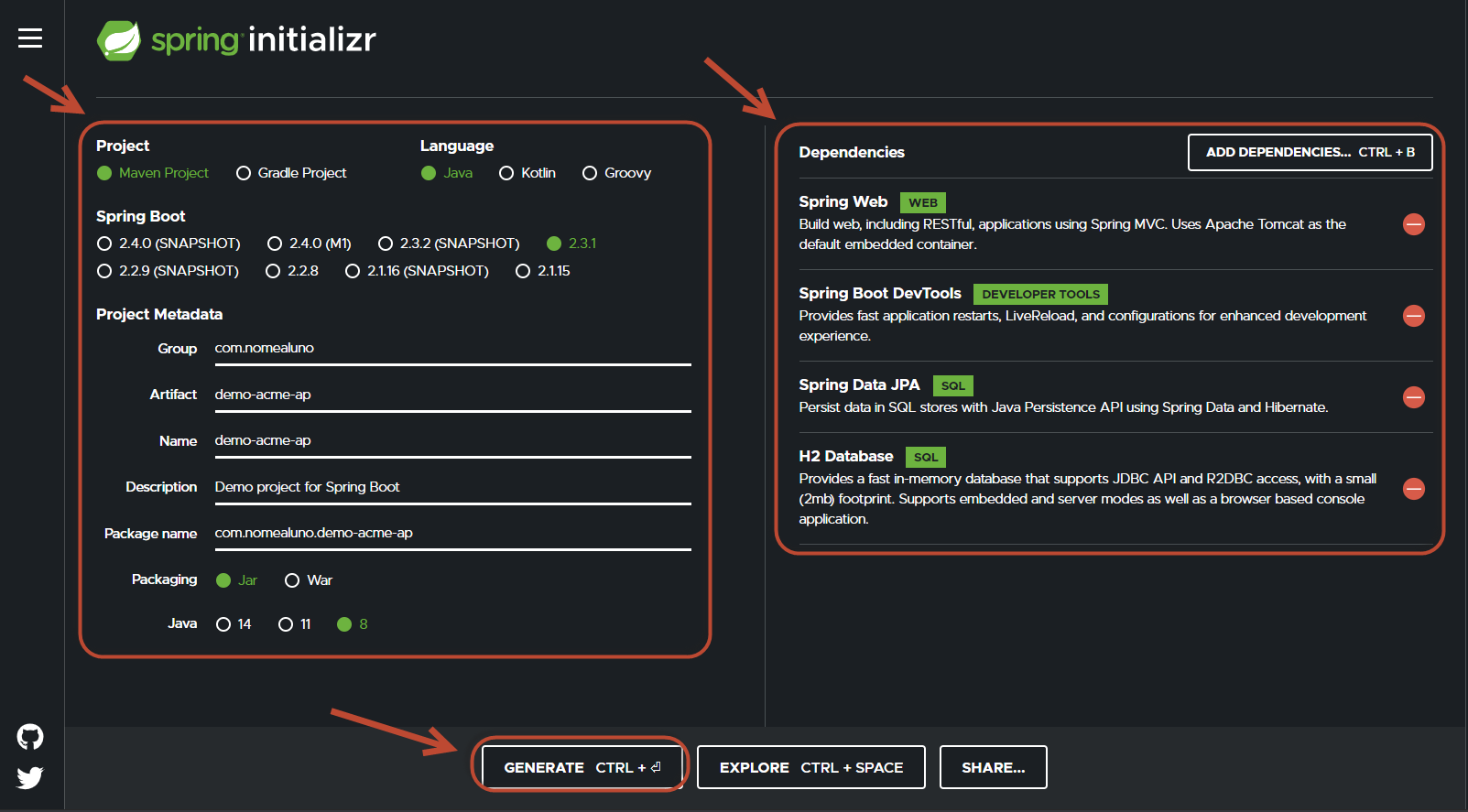
* Descompactar o arquivo que foi baixado em uma pasta do computador e executar a IDE.



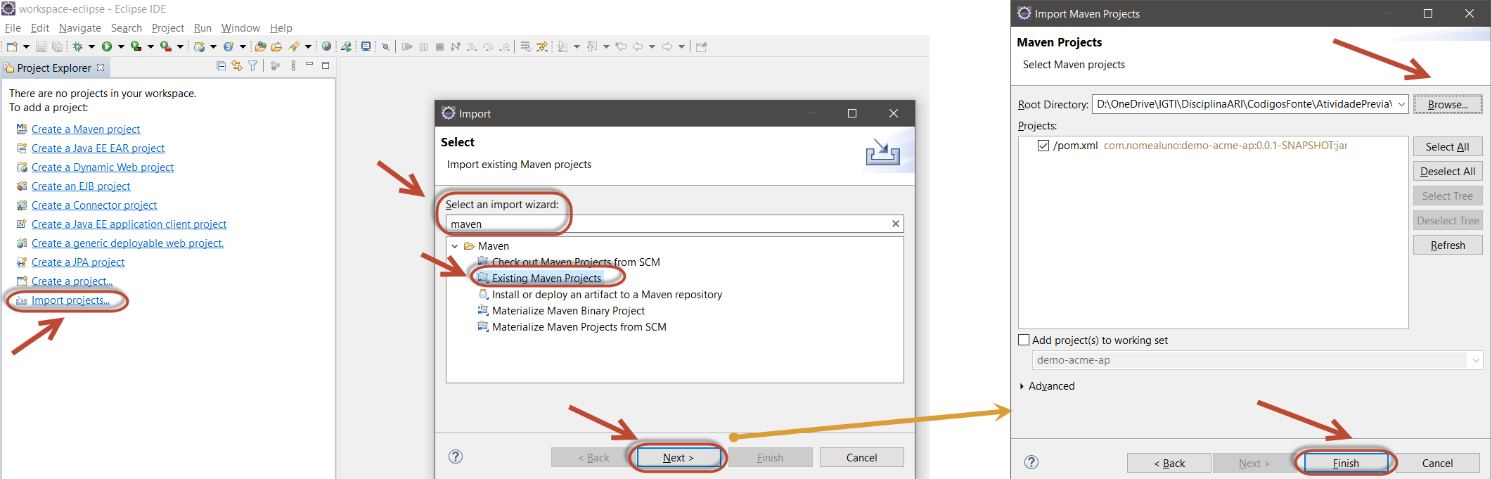


1. Implementação de uma aplicação monolítica (monolito) para o cenário do estudo de caso.

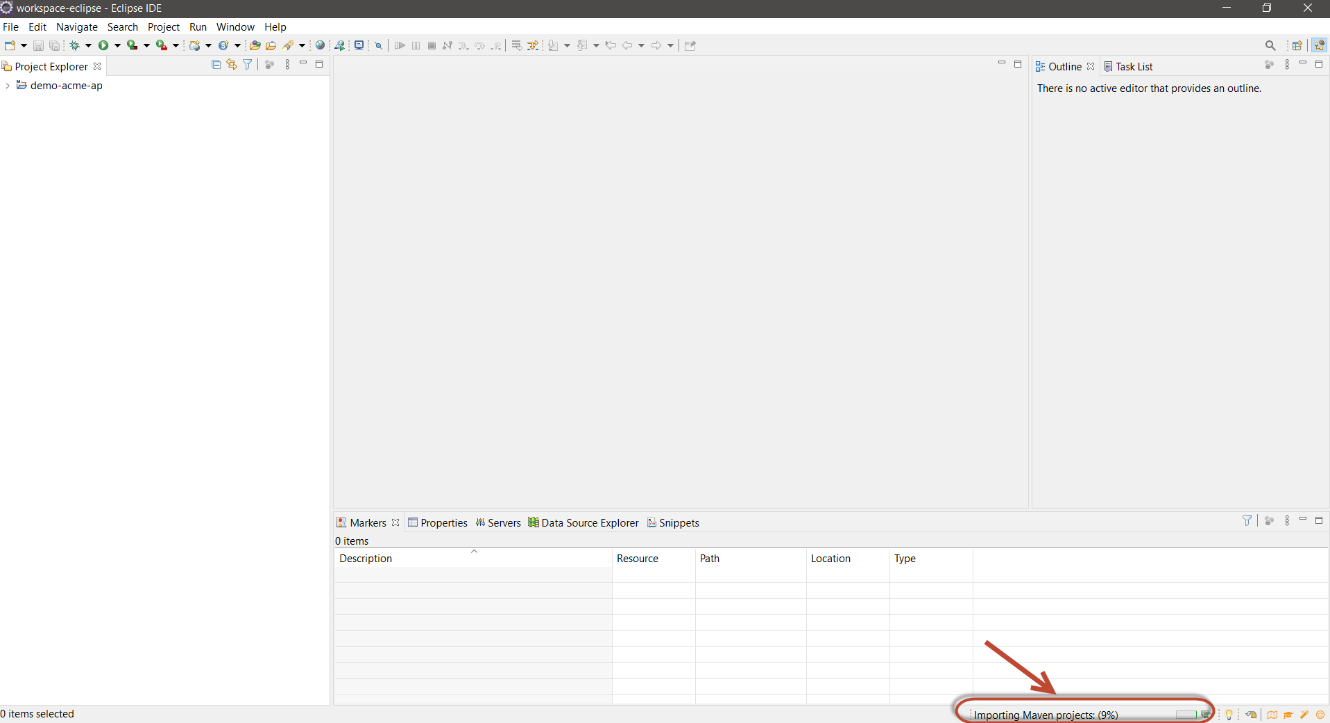
* Acessar o site <https://start.spring.io/>;
* Criar pacote de definição da aplicação monolítica adicionando os pacotes para WEB, DevTools, JPA, banco de dados H2;
* No campo Group, utilizar com.nomealuno, onde nomealuno deve ser trocado pelo primeiro e último nome juntos do aluno(a);
* Realizar o download da estrutura básica da solução e descompactar em uma pasta.



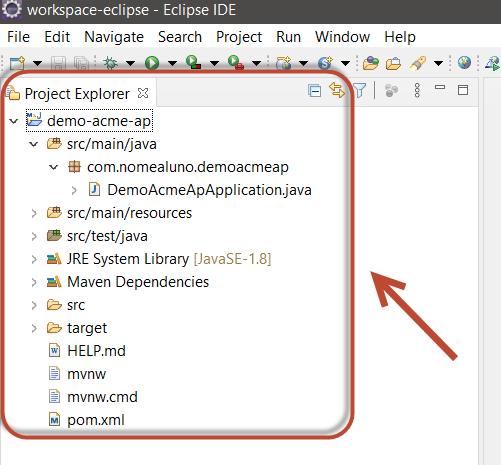
* Abrir novamente a IDE Eclipse e importar o projeto criado anteriormente acessando a pasta onde o arquivo com a estrutura do projeto foi descompactada.



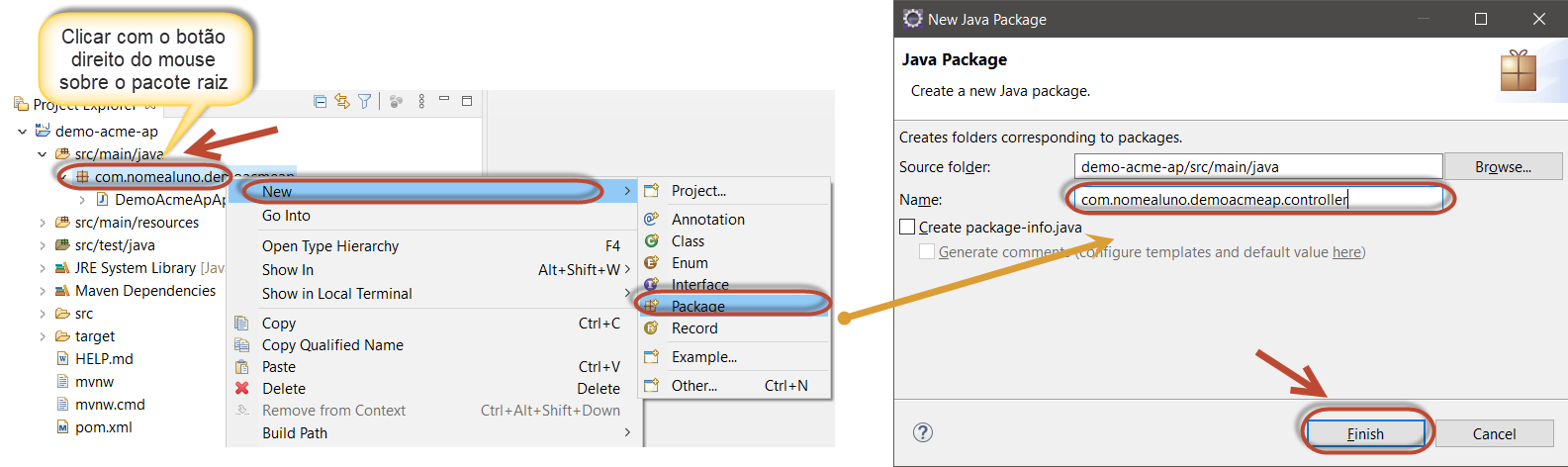
* Aguardar a solução (Maven) atualizar a lista de pacotes de dependencia necessários a solução.



* Após a atualização das dependências, a solução deve possuir a estrutura abaixo:

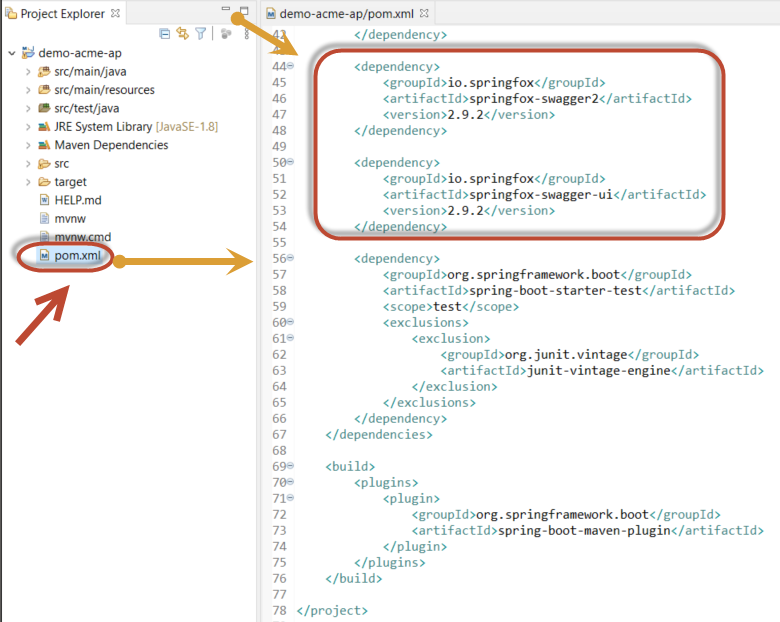


* Criar a seguinte estrutura de pacotes (controller, domain, repositor e exception) na aplicação monolítica, repetindo o processo abaixo para criar cada pacote:



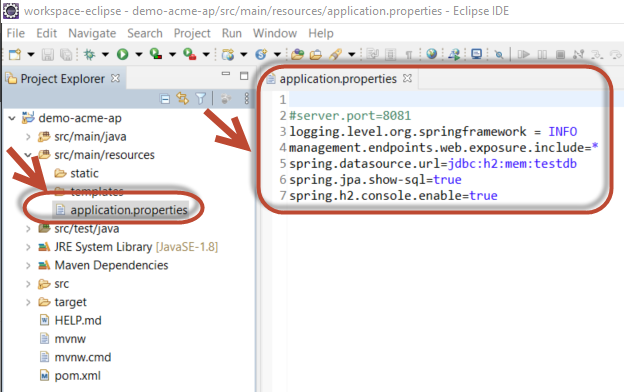
* Como utilizaremos o Swagger para facilitar documentação, localização e acesso aos serviços, abra o arquivo denominado *pom.xml*, que fica na raiz do projeto, e acrescente as dependências abaixo:

Texto para copiar e colar: <https://github.com/profdiovani/mbari/blob/master/pom.xml>



* Como utilizaremos um banco de dados local em memória (H2), abra o arquivo denominado *application.properties,* que fica na raiz do projeto, e acrescente as dependências abaixo:

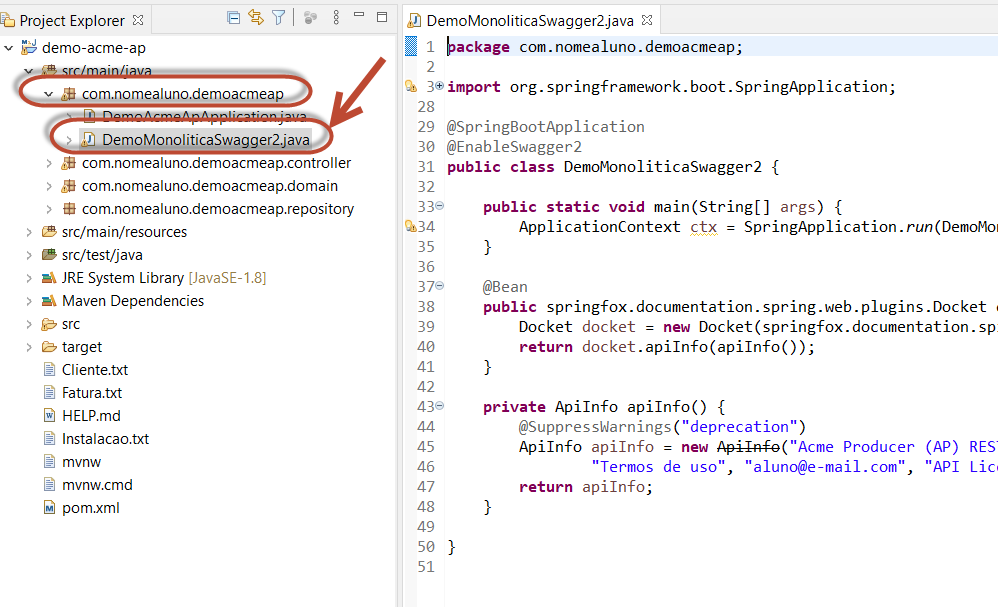
Texto para copiar e colar: <https://github.com/profdiovani/mbari/blob/master/src/main/resources/application.properties>



* Acrescentar as classes em seus respectivos pacotes da solução:

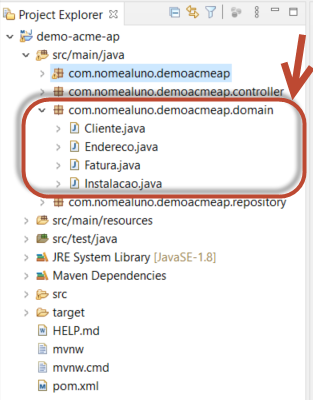
Pacote: com.nomealuno.demoacmeap

Texto para copiar e colar: <https://github.com/profdiovani/mbari/blob/master/src/main/java/com/nomealuno/demoacmeap/DemoMonoliticaSwagger2.java>



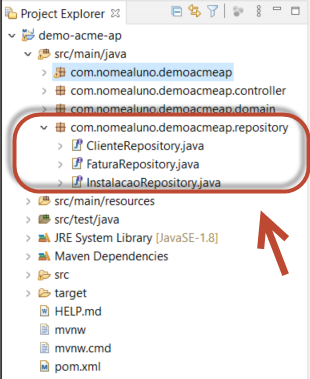
Pacote: com.nomealuno.demoacmeap.domain

Texto para copiar e colar: <https://github.com/profdiovani/mbari/tree/master/src/main/java/com/nomealuno/demoacmeap/domain>



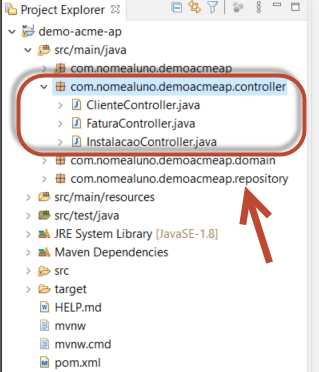
Pacote: com.nomealuno.demoacmeap.repository

Texto para copiar e colar: <https://github.com/profdiovani/mbari/tree/master/src/main/java/com/nomealuno/demoacmeap/repository>



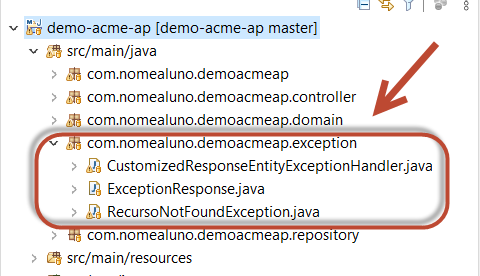
Pacote: com.nomealuno.demoacmeap.controller

Texto para copiar e colar: <https://github.com/profdiovani/mbari/tree/master/src/main/java/com/nomealuno/demoacmeap/controller>

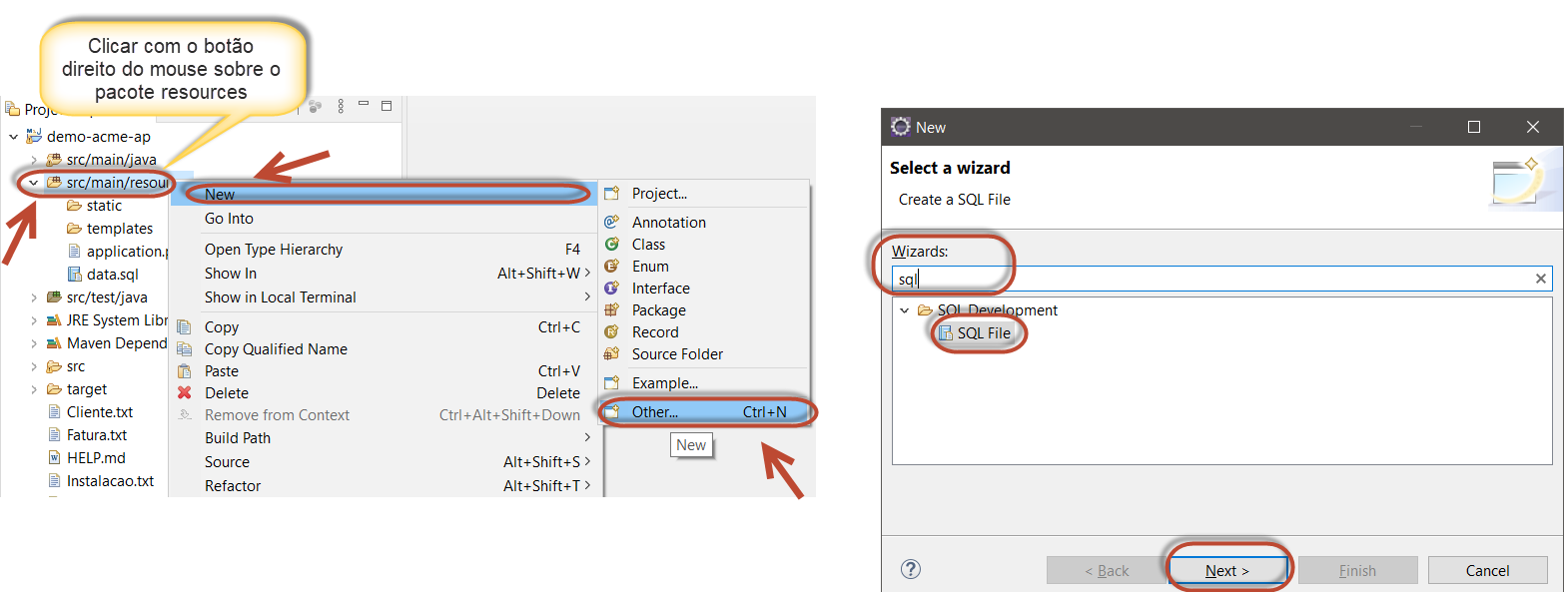


Pacote: com.nomealuno.demoacmeap.exception

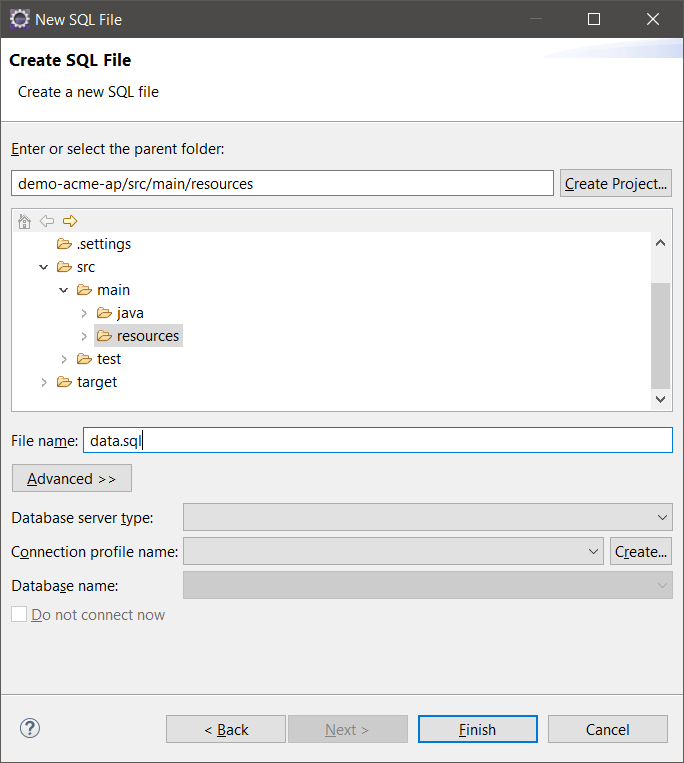
Texto para copiar e colar: <https://github.com/profdiovani/mbari/tree/master/src/main/java/com/nomealuno/demoacmeap/exception>



* Criar arquivo para geração de massa de dados de teste:

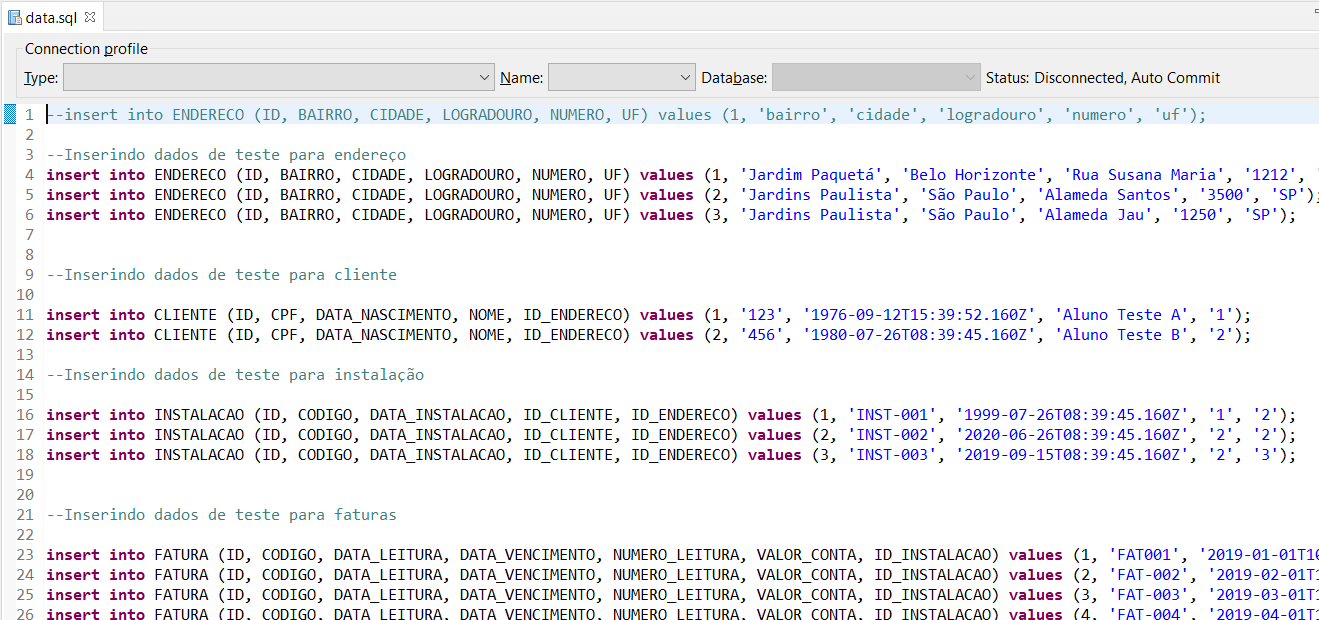


* Informar o nome do arquivo *data.sql.*



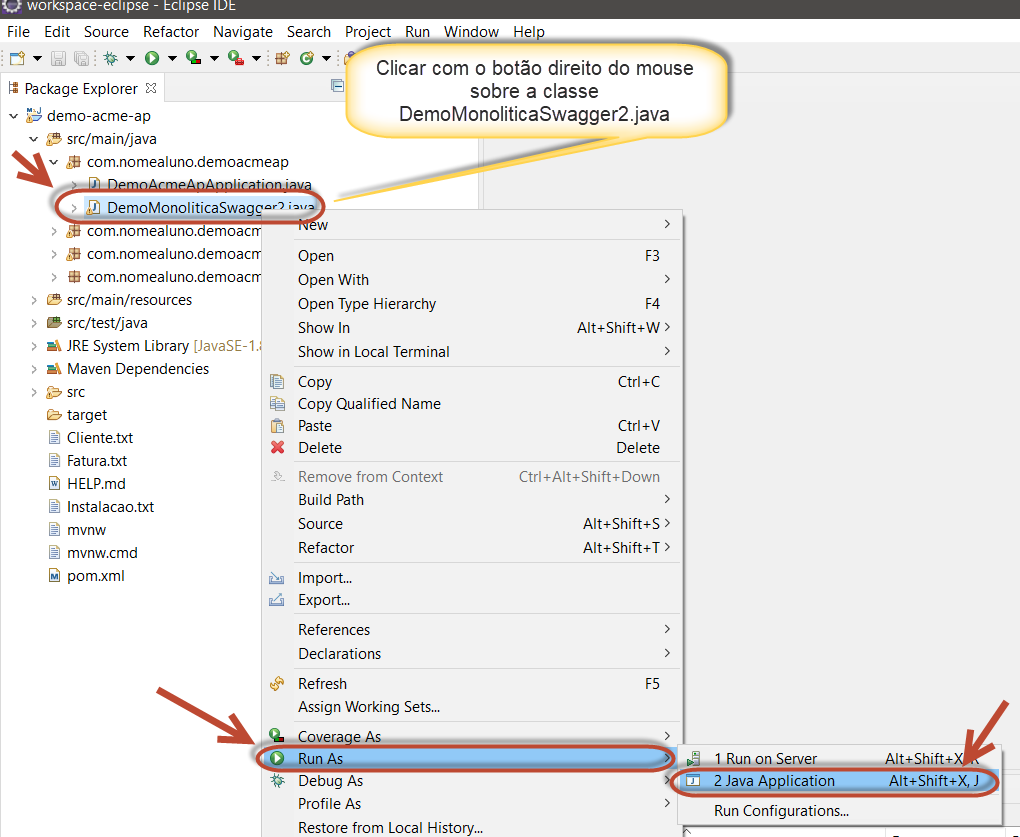
* Gerar dados de teste:

Texto para copiar e colar: <https://github.com/profdiovani/mbari/blob/master/src/main/resources/data.sql>

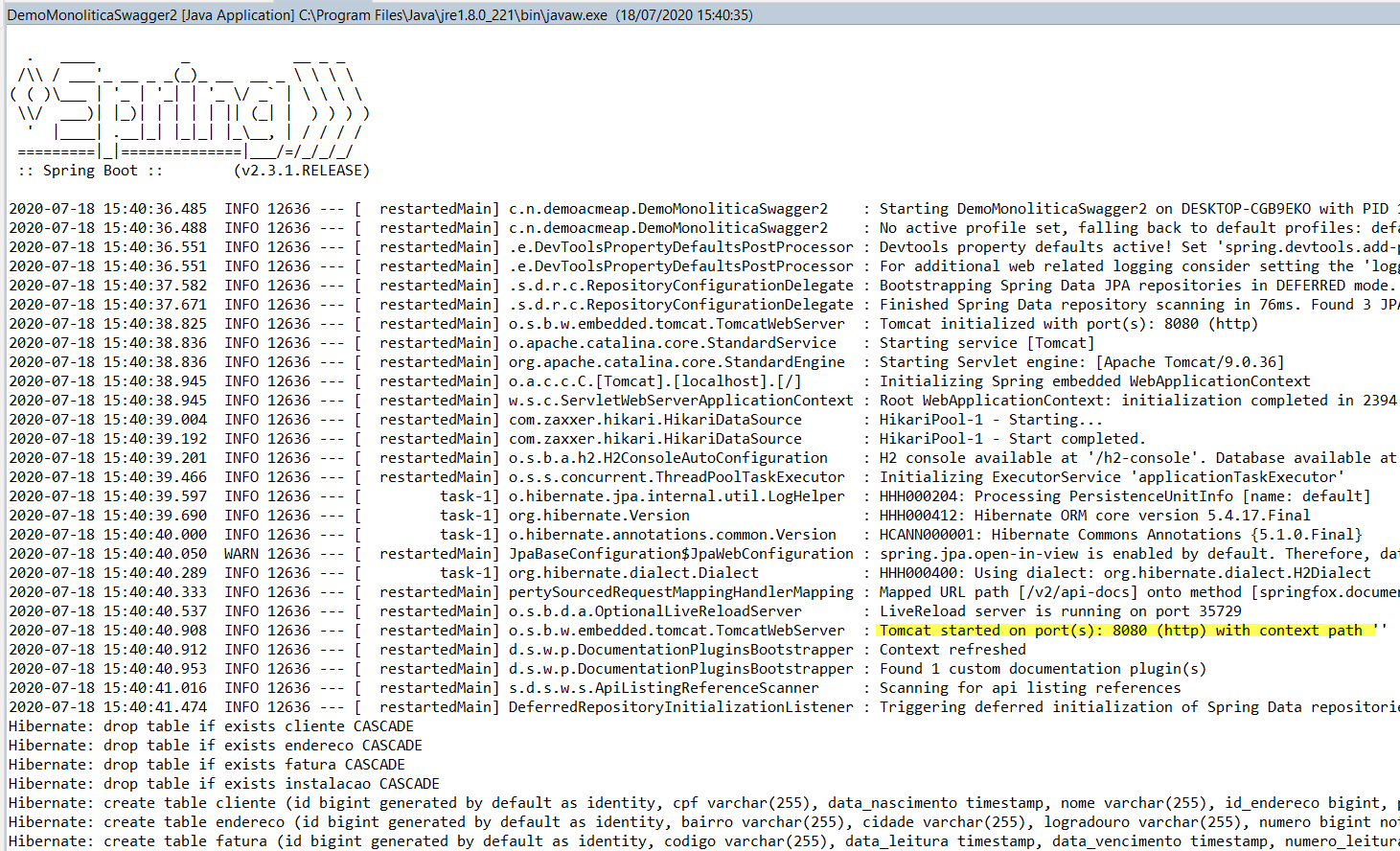


1. Executar e testar a aplicação monolítica.

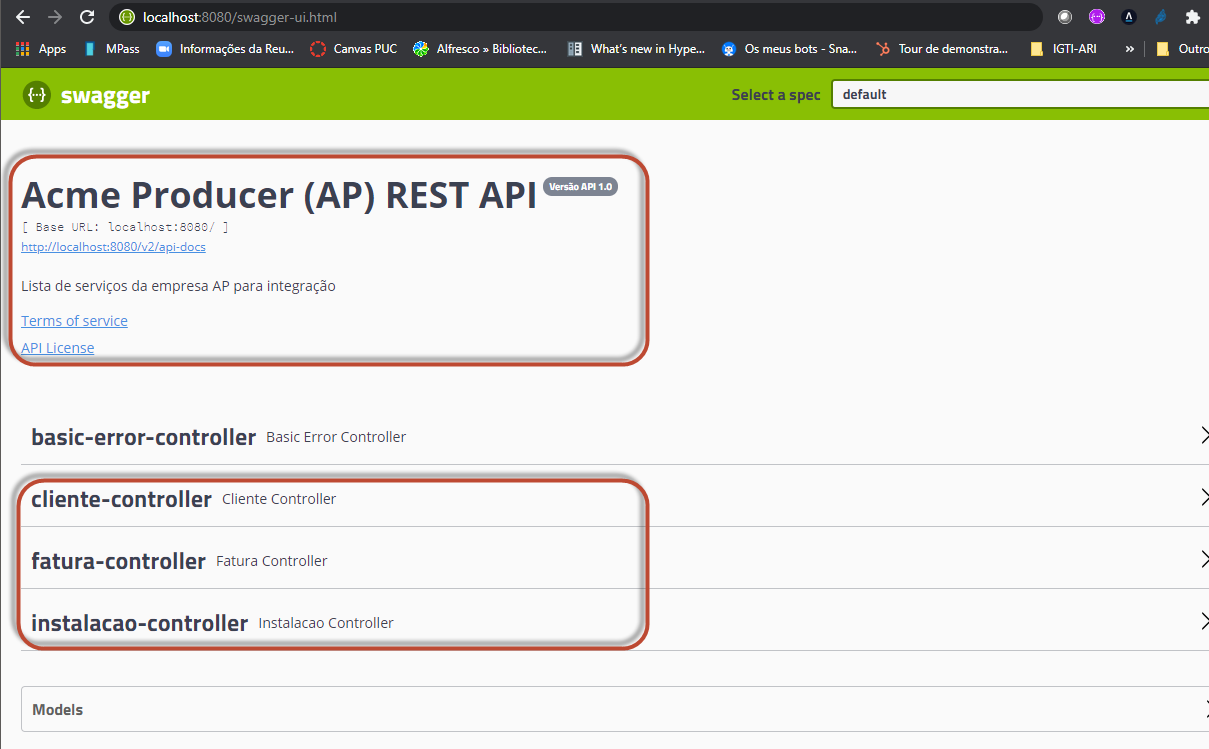
* A execução da aplicação é realizada localmente na própria IDE do Eclipse. Executar a aplicação conforme abaixo:



* Se a execução estiver ocorrido corretamente e nenhum erro exista nos códigos fontes da aplicação, a imagem abaixo deve ser exibida:



* Acessar o link da aplicação: <http://localhost:8080/swagger-ui.html>



* Navegar nos serviços e executar:



* Para inclusão de um cliente, utilizar a entrada abaixo:

Link para copiar e colar: <https://github.com/profdiovani/mbari/blob/master/Cliente.txt>

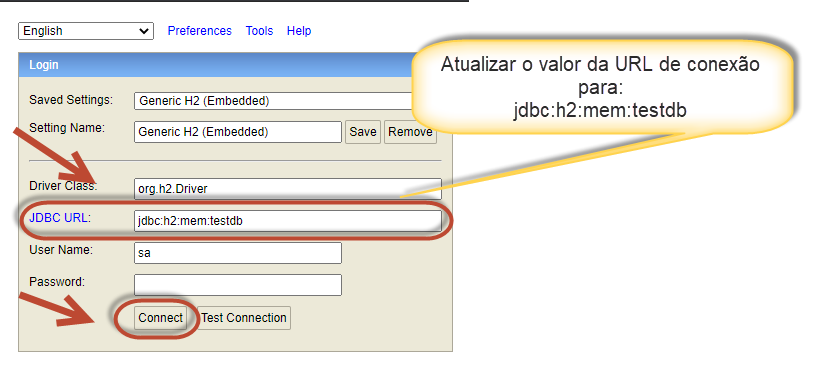
* Para inclusão de uma instalação utilizar a entrada abaixo:

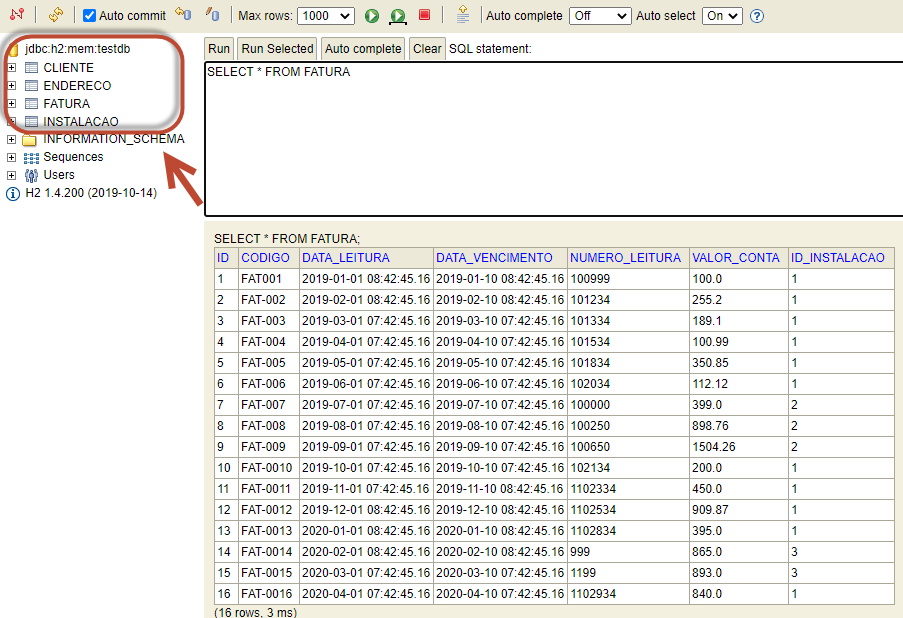
Link para copiar e colar: <https://github.com/profdiovani/mbari/blob/master/Instalacao.txt>

* Para gerar uma fatura utilizar a entrada abaixo:

Link para copiar e colar: <https://github.com/profdiovani/mbari/blob/master/Fatura.txt>

* Para acessar o banco de dados local em memória utilizar o link: <http://localhost:8080/h2-console/>



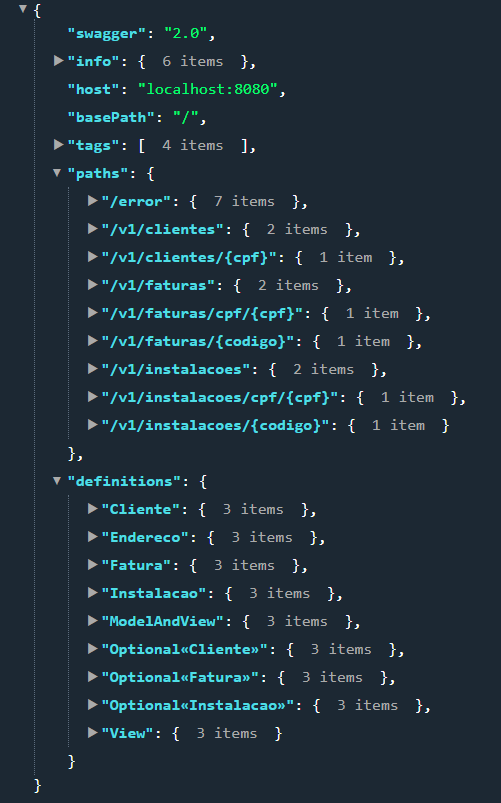


1. Para os alunos que tiverem alguma dificuldade para realização dos passos anteriores (Passo 2), acessar o link abaixo e fazer uma cópia completa da solução para realizar a análise crítica:

* Link para download: <https://github.com/profdiovani/mbari>

1. Fazer uma análise crítica sobre a aplicação monolítica, destacando o entendimento sobre o seu funcionamento, quais seriam os benefícios e/ou problemas dessa aplicação em uma arquitetura de integração com clientes.

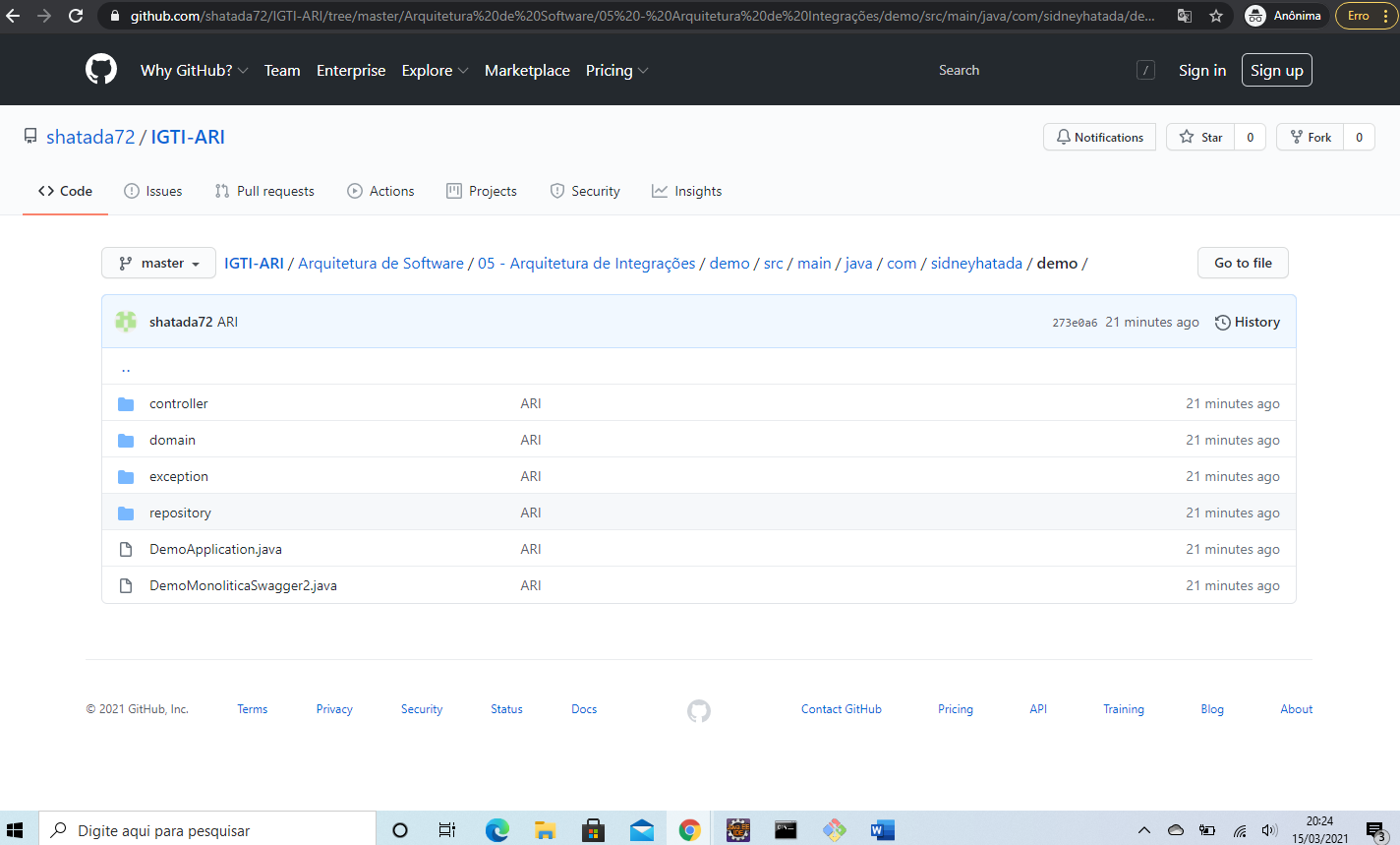




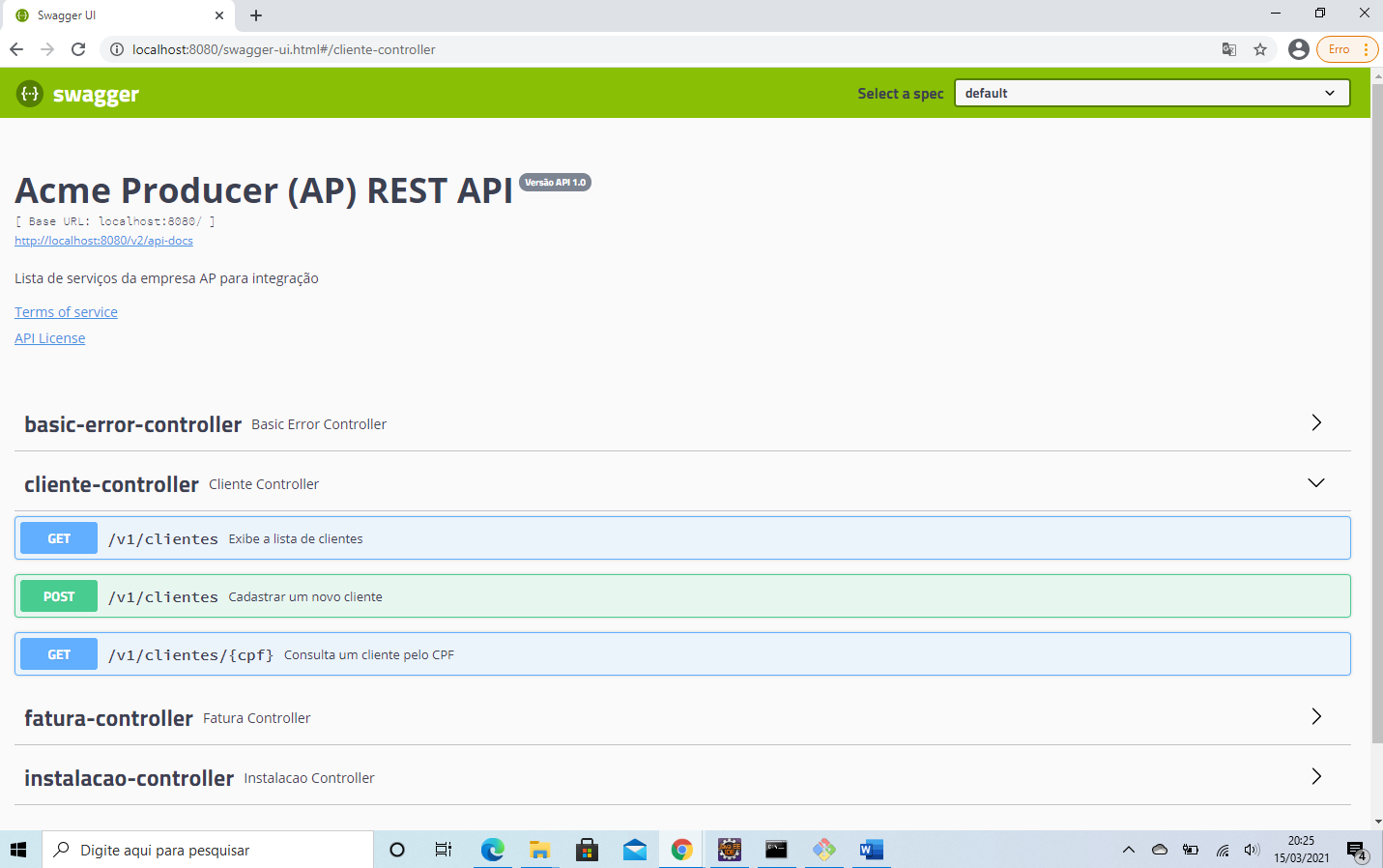
**Análise crítica da arquitetura da solução**

A solução criada está hospedada em:

<https://github.com/shatada72/IGTI-ARI/tree/master/Arquitetura%20de%20Software/05%20-%20Arquitetura%20de%20Integra%C3%A7%C3%B5es/demo/src/main/java/com/sidneyhatada/demo>



Apresentação da solução funcionando:



Análise:

A aplicação criada é em uma arquitetura monolítica, isto é, formam um único sistema e possuem módulos separados que fazem funções distintas, porém fazendo parte de um mesmo deploy, sendo que a cada alteração do código, devemos subir todos os módulos juntos.

Podemos citar como vantagens:

* Facilidade no desenvolvimento pois é um único sistema
* Facilidade no teste
* Facilidade de deploy para o servidor
* Facilidade de escalar

Como desvantagens temos:

* Manutenção, uma vez que com o crescimento da aplicação, será mais difícil o entendimento do código e a sua alteração
* Alteração complicada, pois é necessário fazer um novo deploy a cada alteração
* Linha de código mal escrito pode impactar todo o ambiente
* Falta de flexibilidade na linguagem de programação, uma vez que se foi iniciada em uma linguagem, deverá ser a padrão.

Muitos sistemas já estão sendo desenvolvidos na Arquitetura de Microsserviços e outros estão sendo alterados, o qual prevê algumas flexibilidades a mais, porém o framework a ser adotado deverá levar em consideração aspectos como:

* Expertise da equipe
* Maturidade da Empresa
* Custo / Benefício da solução
* Crescimento da Solução
* Demora no Deploy
* Dificuldade na manutenção
* etc