

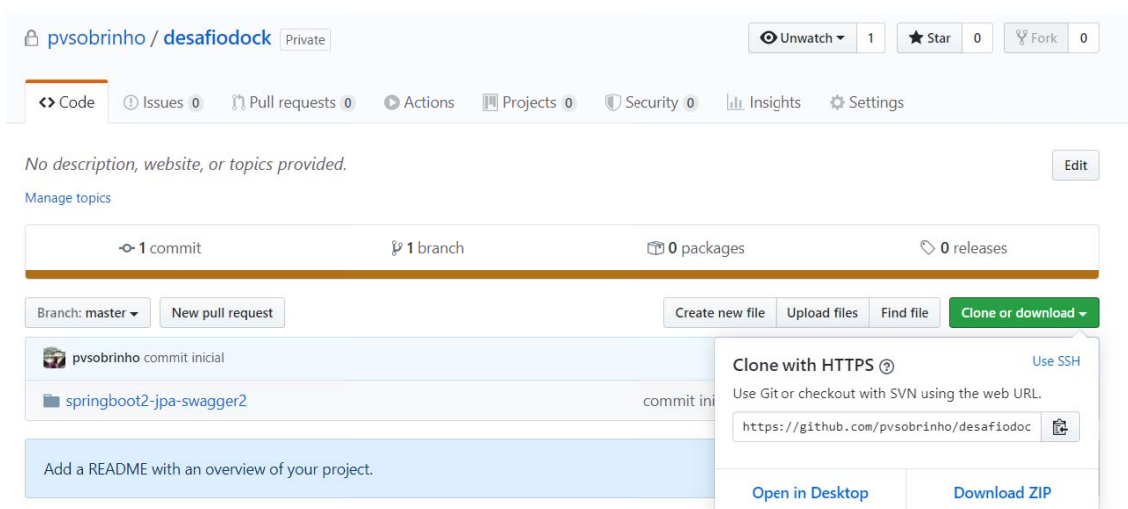


Conteúdo deste Manual

- (1) Baixar o projeto
- (2) Configurações na IDE e JDK
- (3) Configurando o Banco de Dados
- (4) Script de Banco de Dados
- (5) Teste da aplicação com Swagger

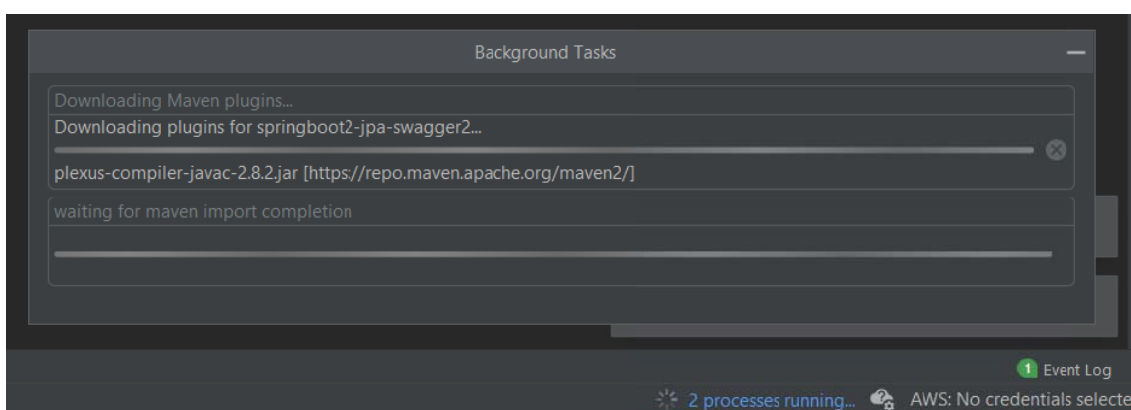
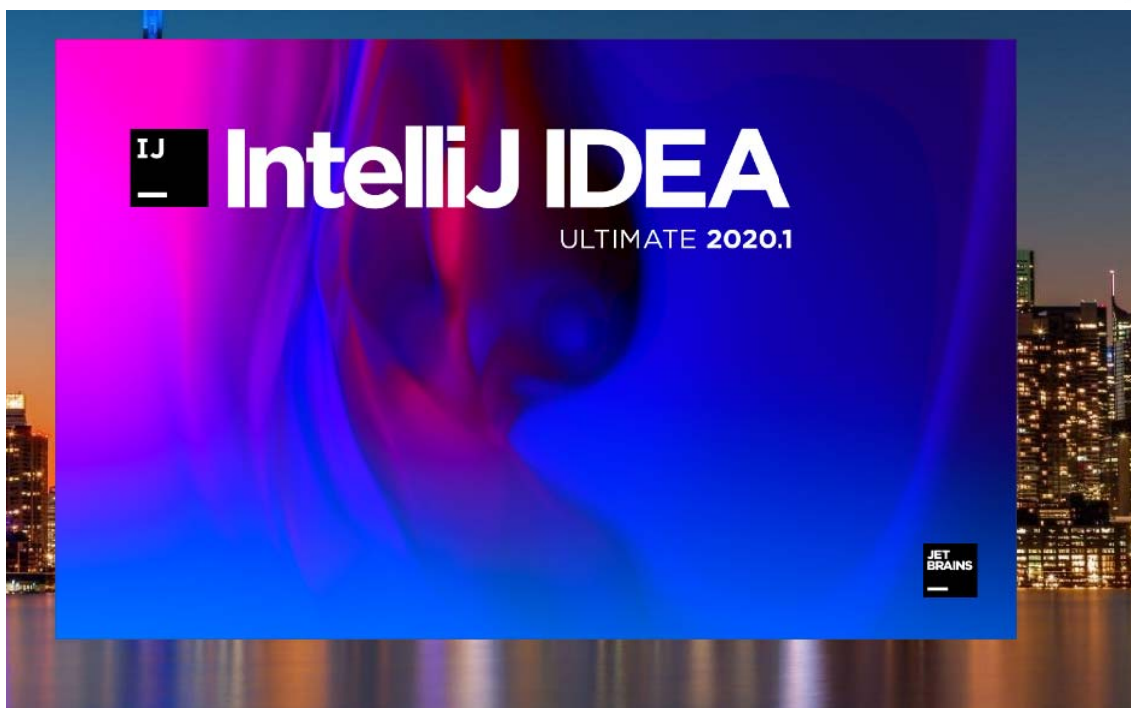
1- Baixar o projeto

Realizar o Clone do Projeto no GitHub em: <https://github.com/pvsobrinho/desafiodock.git>

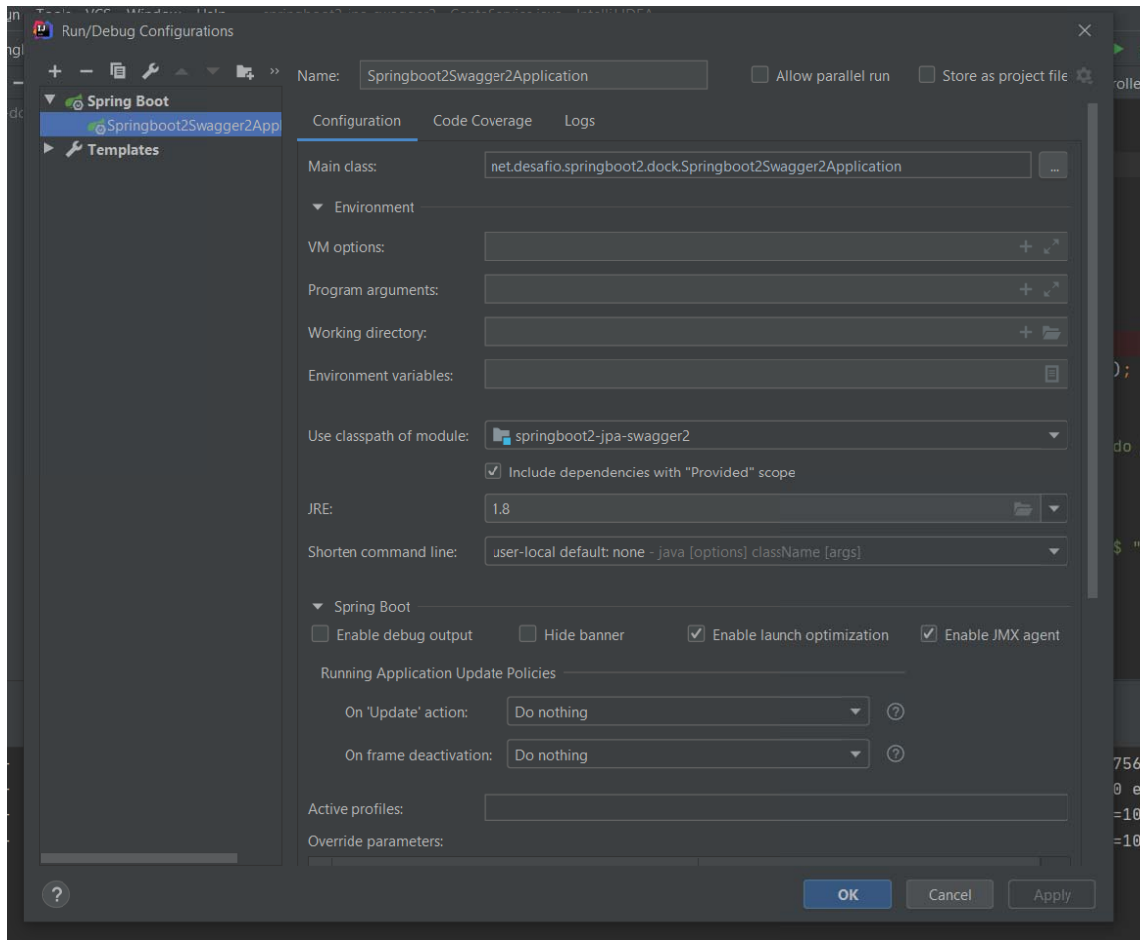


2- Configuração na IDE e JDK

Abrir o projeto preferencialmente no IntelliJ a IDE para automatizar operações de downloads de dependências e build e configuração automática do Firewall do seu sistema operacional. Pode utilizar outra IDE de sua preferência caso deseje.



Foi utilizado esta IDE por conta da facilidade de configuração e algumas ações que são feitas automaticamente. Porém é necessário ficar atento a versão do JDK compatível com o projeto. JDK 8.



3- Configurando o Banco de Dados

Configuração do modelo para criação da base de dados:

Type and Networking

Server Configuration Type

Choose the correct server configuration type for this MySQL Server installation. This setting will define how much system resources are assigned to the MySQL Server instance.

Config Type:

Connectivity

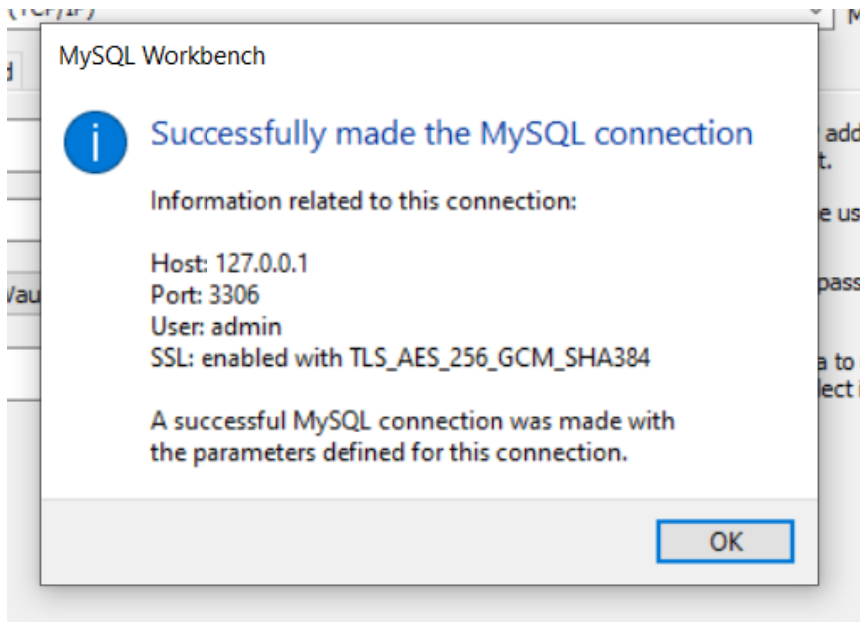
Use the following controls to select how you would like to connect to this server.

- ☒ TCP/IP Port: X Protocol Port:
- ☒ Open Windows Firewall ports for network access
- ☐ Named Pipe Pipe Name:
- ☐ Shared Memory Memory Name:

Advanced Configuration

Select the check box below to get additional configuration pages where you can set advanced and logging options for this server instance.

- ☐ Show Advanced and Logging Options



Na aplicação procurar o arquivo `application.properties` e mudar a url de conexão e as credenciais para acessar o banco de dados. Para este exemplo foi utilizado a base de dados com nome ***schema = dock***.

```
## Spring DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url = jdbc:mysql://localhost:3306/dock?useSSL=false
## 127.0.0.1 = localhost
spring.datasource.username = admin
spring.datasource.password = admin

spring.datasource.testWhileIdle = true
spring.datasource.validationQuery = SELECT 1
spring.jpa.show-sql = true

## Hibernate Properties
# The SQL dialect makes Hibernate generate better SQL for the chosen database
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5InnoDBDialect

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update

logging.level.root = DEBUG
```



4. Script de Banco de Dados

SCRIPT DE CRIACAO DO BANCO DE DADOS

//CREATE SCHEMA

```
CREATE SCHEMA `dock` ;
```

SCRIPT DE CRIACAO DA TABELA CONTA

O SPRINGBOOT CRIA AUTOMATICAMENTE A TABELA CASO ELA NÃO EXISTA NO BANCO DE DADOS, MAS CASO QUEIRA CRIAR MANUALMENTE SEGUE O SCRIPT

■ CONTA

```
SELECT * FROM dock.conta;CREATE TABLE `conta` (  
  
  `id_conta` bigint NOT NULL,  
  
  `data_criacao` datetime NOT NULL,  
  
  `flag_ativo` bigint NOT NULL,  
  
  `id_pessoa` bigint NOT NULL,  
  
  `limite_saque_diario` decimal(19,2) NOT NULL,  
  
  `saldo` decimal(19,2) NOT NULL,  
  
  `tipo_conta` bigint NOT NULL,  
  
  PRIMARY KEY (`id_conta`)  
  
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;
```

■ PESSOA

```
CREATE TABLE `pessoa` (  
  
  `id_pessoa` bigint NOT NULL,  
  
  `cpf` varchar(255) NOT NULL,  
  
  `data_nascimento` datetime NOT NULL,  
  
  `nome` varchar(255) NOT NULL,
```



PRIMARY KEY (`id_pessoa`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

■ PESSOA_CONTA

CREATE TABLE `pessoa_conta` (

`id_pessoa_conta` bigint NOT NULL,

`id_conta` bigint NOT NULL,

`id_pessoa` bigint NOT NULL,

PRIMARY KEY (`id_pessoa_conta`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

SELECT * FROM dock.pessoa_conta;

■ TRANSACAO

CREATE TABLE `transacao` (

`id_transacao` bigint NOT NULL,

`data_transacao` datetime NOT NULL,

`descricao` varchar(255) NOT NULL,

`id_conta` bigint NOT NULL,

`valor` decimal(19,2) NOT NULL,

PRIMARY KEY (`id_transacao`)

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci;

-- SCRIPT CRIACAO DE REGISTRO NA TABELA PESSOA

INSERT INTO `dock`.`pessoa` (`id_pessoa`, `cpf`, `data_nascimento`, `nome`) VALUES ('14', '124.456.678-45', '2020-05-20 23:58:01', 'Maria');



5. Teste da aplicação com Swagger

ACESSAR O SWAGGER EM : <http://localhost:8080/swagger-ui.html#/>

Swagger UI interface for the Desafio Dock REST API. The API is titled "Desafio Dock REST API 1.0.0" and has a base URL of "http://localhost:8080/v2/api-docs". The API is described as a "Microserviço para manipulação de dados de conta bancária - REST API" and is developed by "Paulo Victor Sobrinho de Jesus".

The API is organized into two main sections:

- conta-controller** (Operações de conta bancária):
 - PUT** /dock/conta/ativar/numeroConta/{idConta}: Ativar uma conta por idConta
 - PUT** /dock/conta/atualizar/{idConta}: Atualizar conta
 - POST** /dock/conta/criar: Criar conta
 - PUT** /dock/conta/depositoValor/{valorCreditar}/numeroConta/{idConta}: Realiza o deposito em uma conta por idConta
 - PUT** /dock/conta/desativar/numeroConta/{idConta}: Desativar uma conta por idConta
- transacao-controller** (Transacoes de conta bancária):
 - GET** /dock/conta/listar: Visualizar a lista de contas
 - GET** /dock/conta/numeroConta/{idConta}: Obter conta por id - consulta limite de saque e Saldo
 - GET** /dock/conta/saldo/numeroConta/{idConta}: Obter saldo da conta por id - consulta limite de saque e Saldo
 - PUT** /dock/conta/saqueValor/{valorDebitar}/numeroConta/{idConta}: Realiza o saque da conta por idConta

Below the account controller, there is a section for **pessoa-controller** (Operações do cliente):

- POST** /dock/pessoa/criar: Cadastrar Cliente
- GET** /dock/pessoa/idPessoa/{idPessoa}: Obter pessoa por id