Conectar a API ao BD
Tipos de arquitetura

ATIVIDADES DE HOJE

Horário das atividades

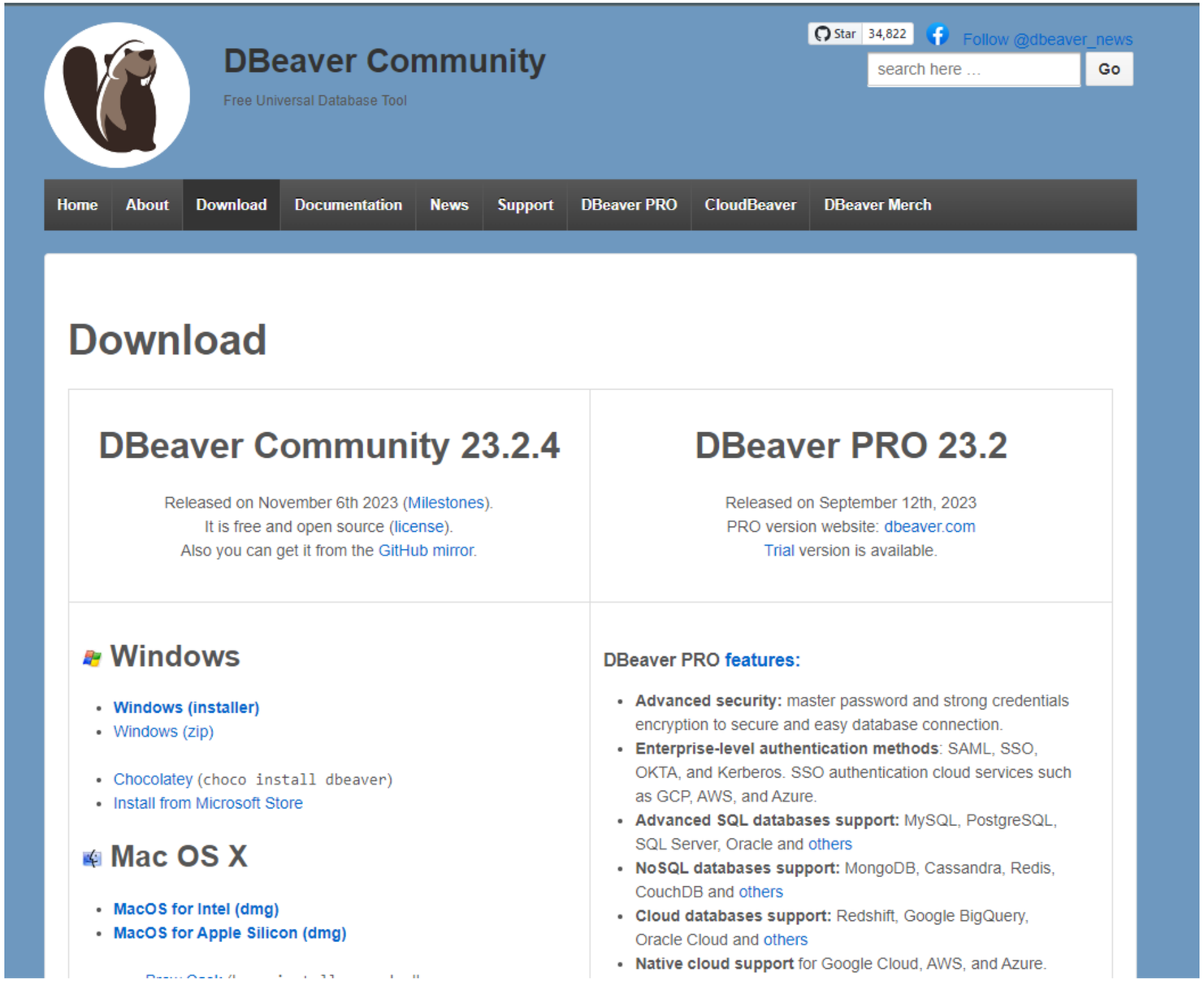
- 13h30 Faísca
- 13h45 Prática SQL
- 15h00 Intervalo
- 15h15 Conectar a API ao BD e Tipos de arquitetura
- 16h40 Checkout
- 17h00 Chamada

Criar o banco

DBeaver

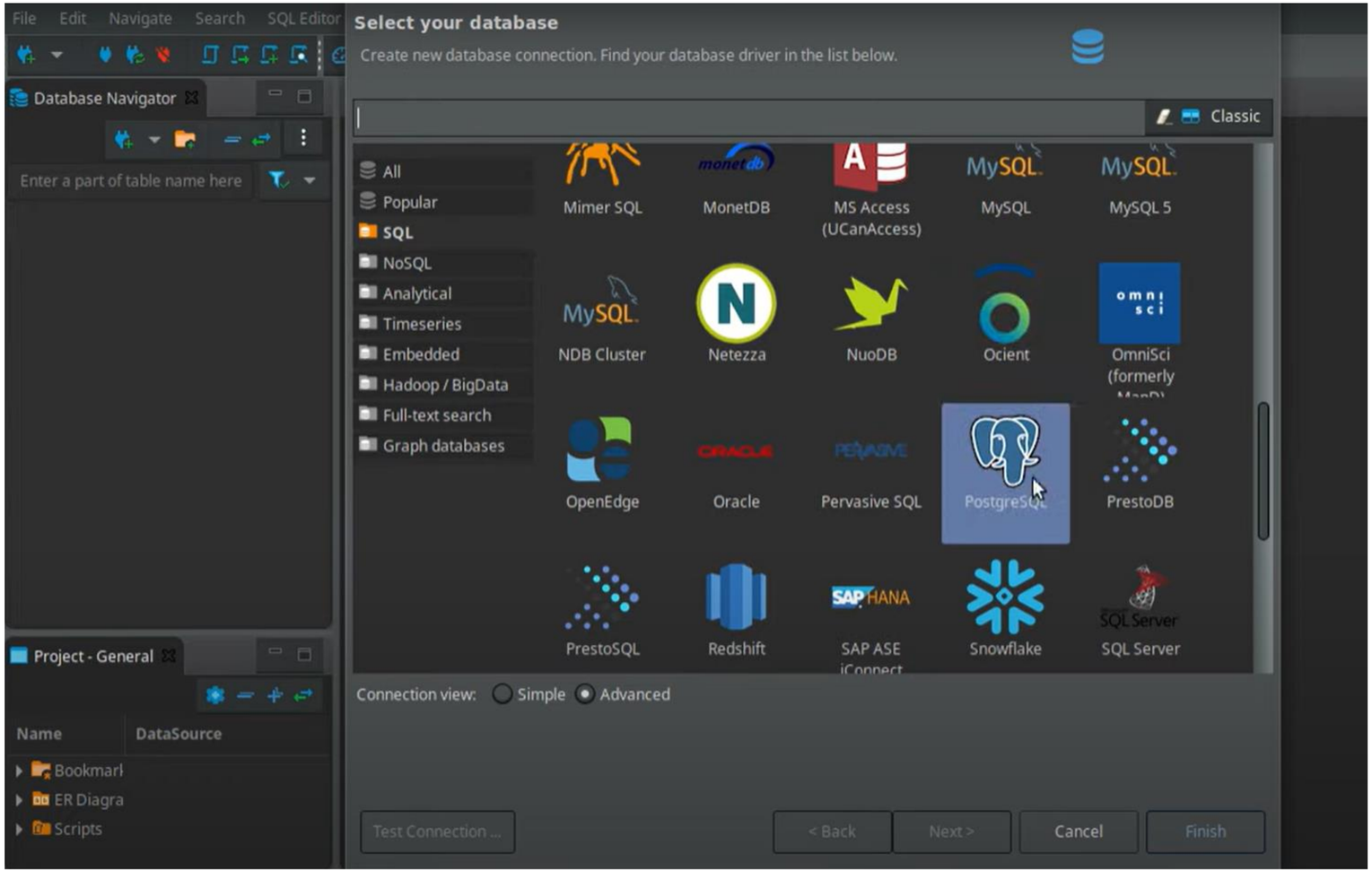
https://dbeaver.io/

- Clique em **Download** e execute;
- Marque o “Associate SQL Script with Database”



Adicionando uma conexão

- Clique na “tomada” no canto superior esquerdo;
- Selecione PostGreSQL;

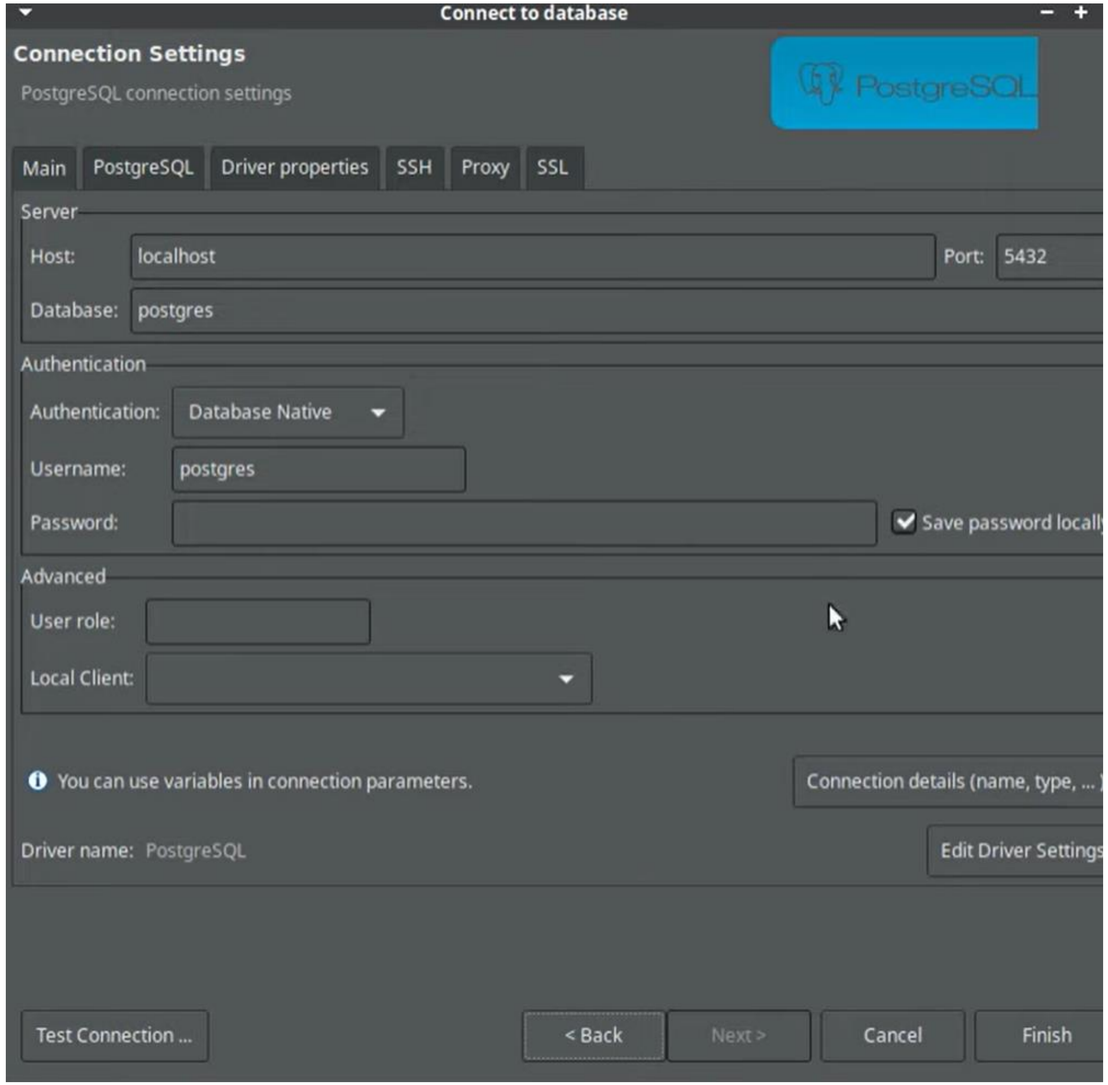


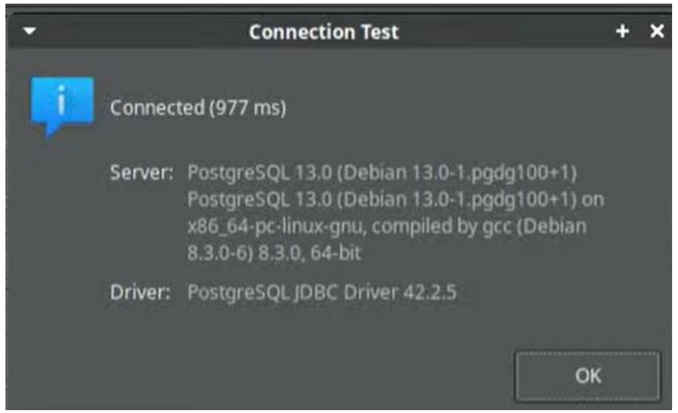
Adicionando uma conexão

- Configure as credenciais do teu banco;

- Database: postgres
- Username: postgres
- Password: postgres

- E teste a conexão;





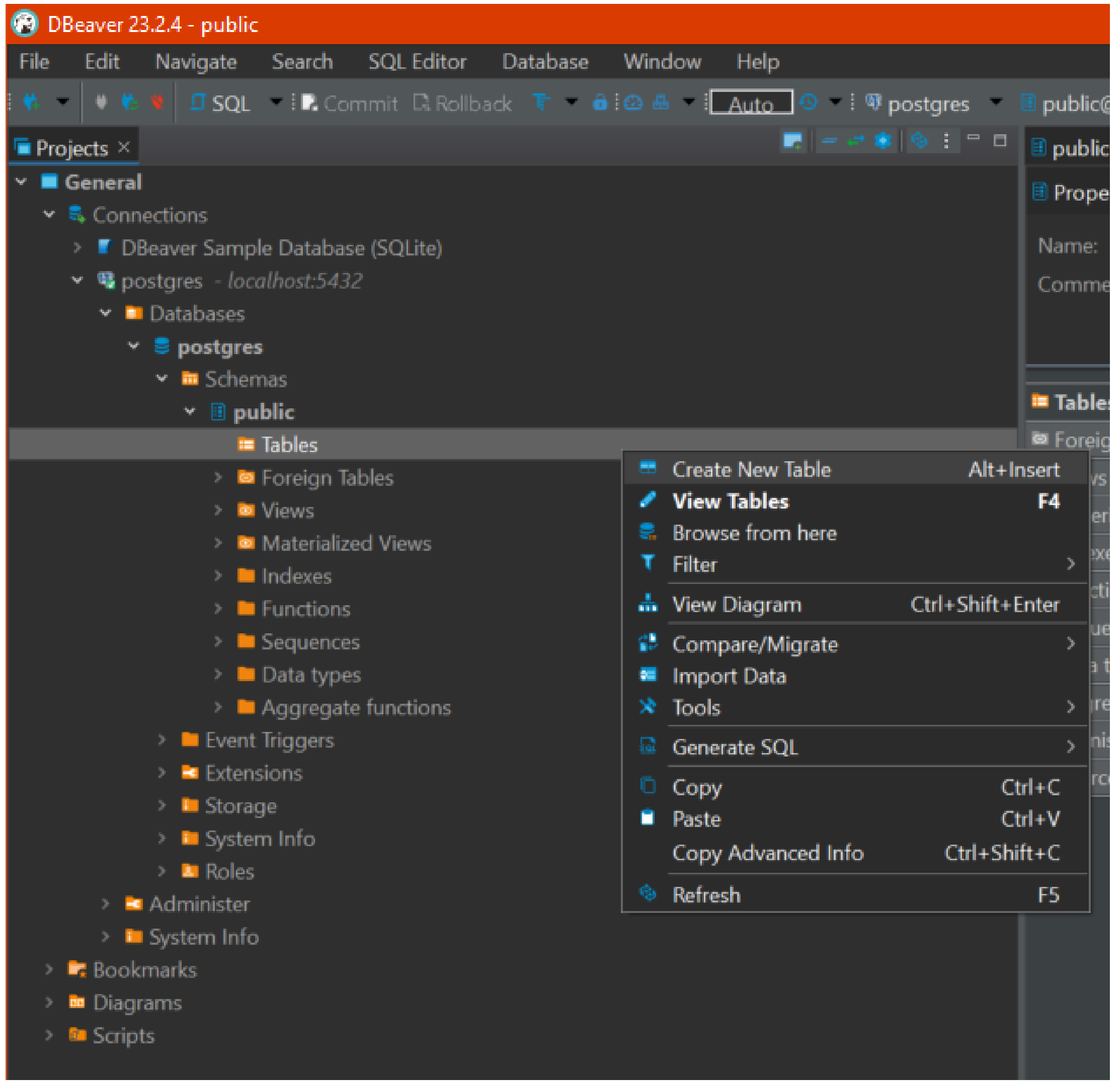
TIC em trilhas

Criando uma table

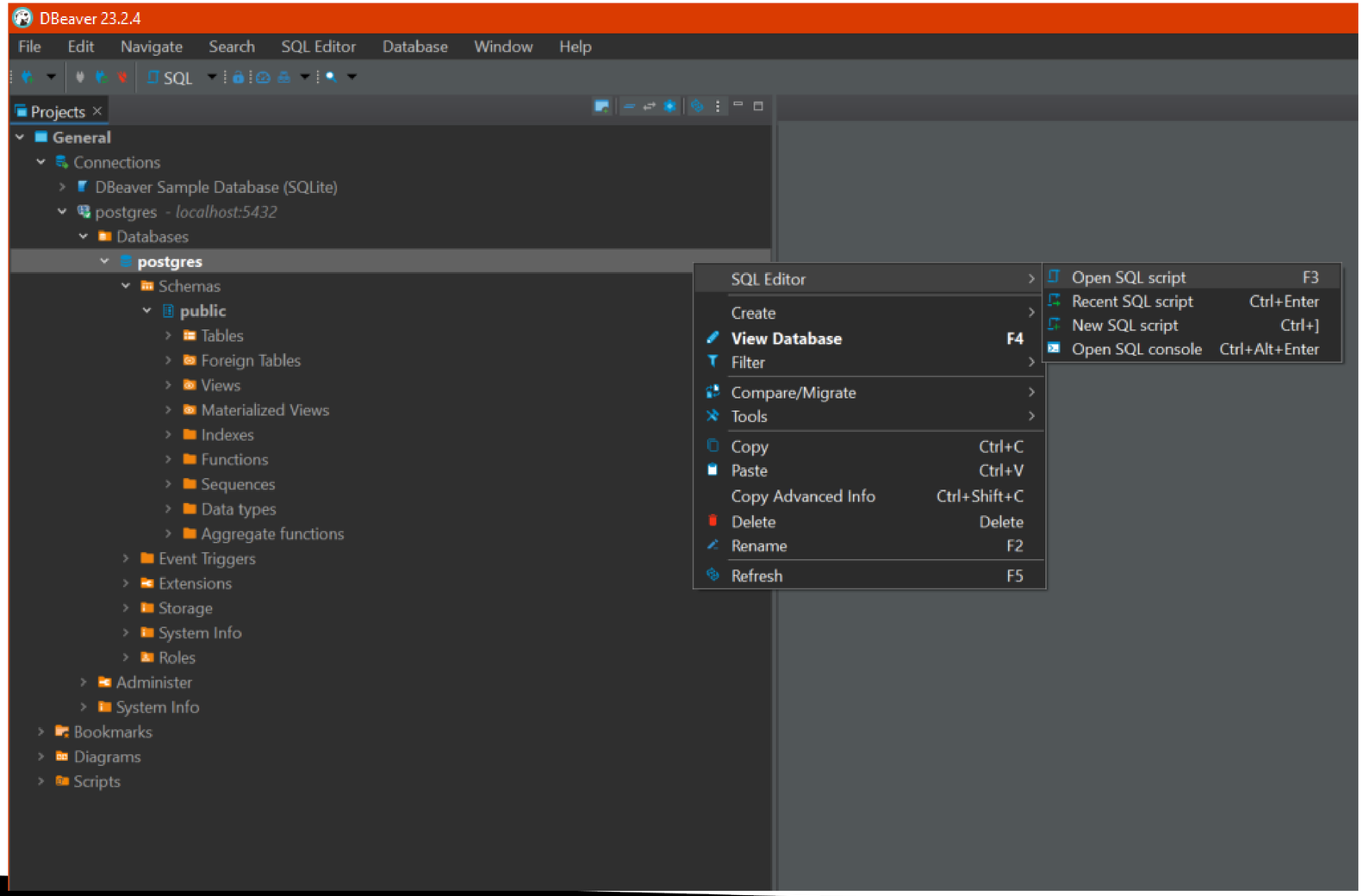
- Siga o caminho na imagem e clique com o botão direito e em **Create New Table**

- Database: postgres
- Username: postgres
- Password: postgres

- E teste a conexão;



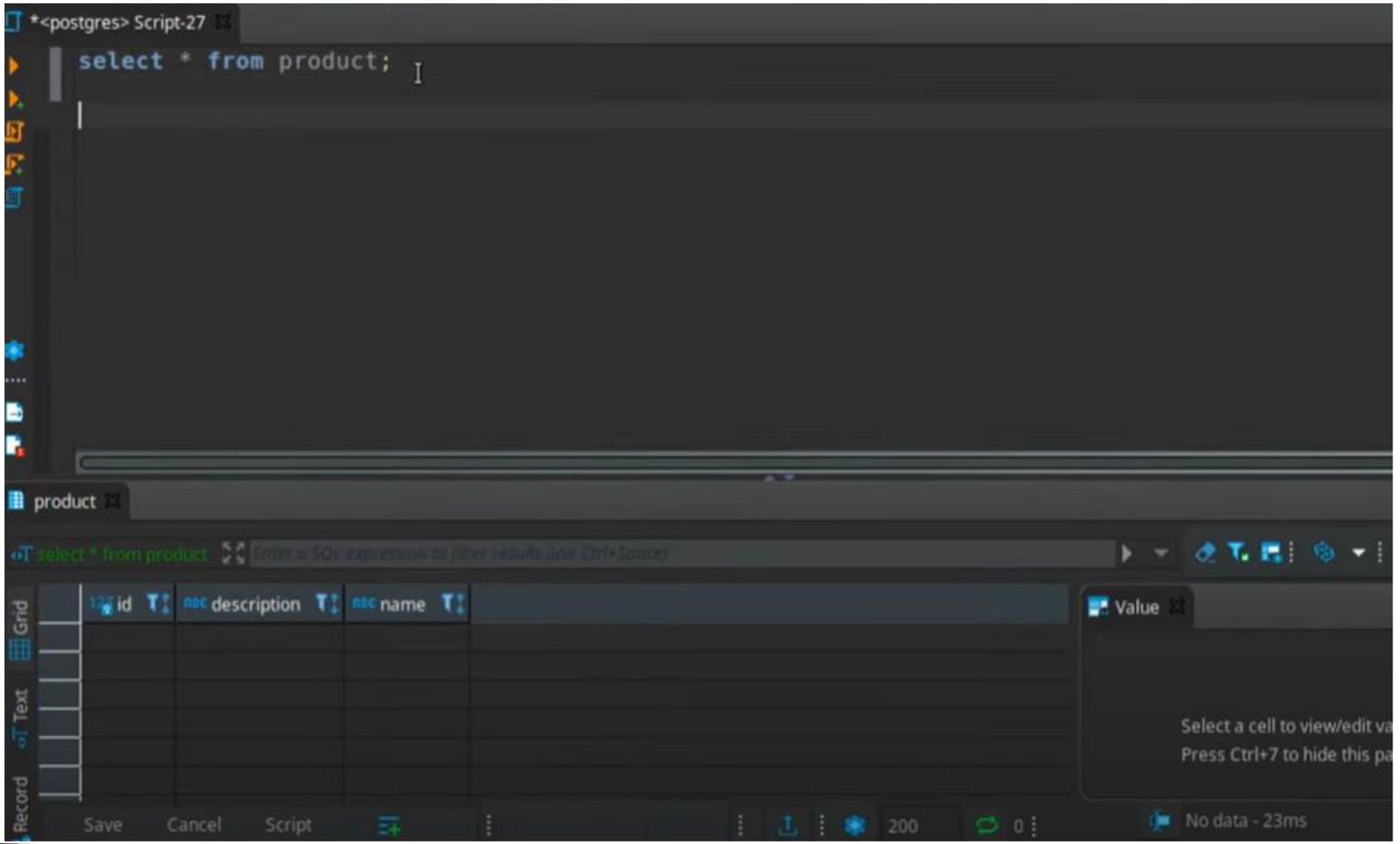
Criando uma querie



TIC em trilhas

Criando uma table

Você pode rodar o código com **CRTL + Enter**



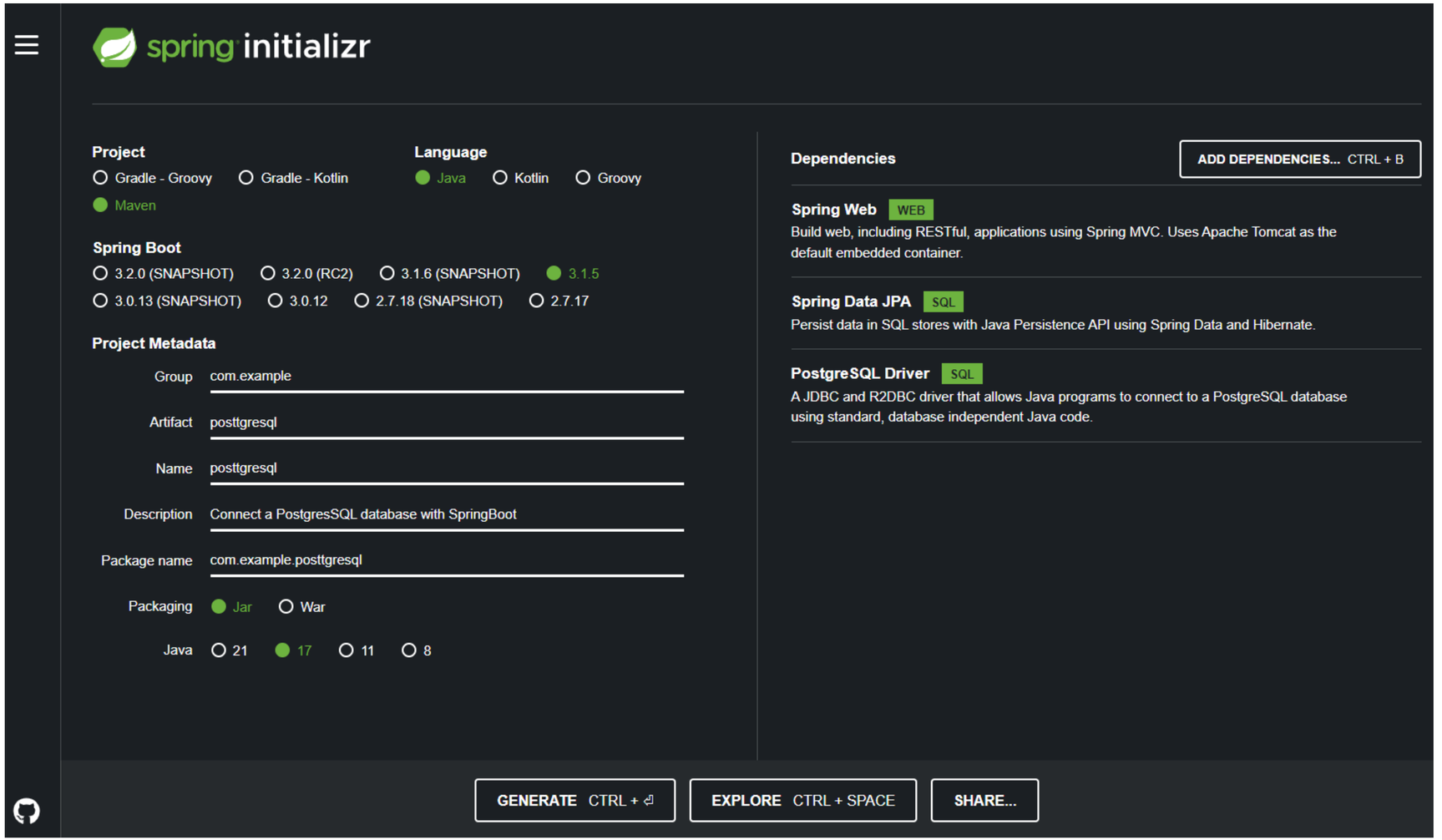
TIC em trilhas

Conectar a API ao BD

TIC em trilhas

Conectar a API ao BD

Criação de um projeto Spring

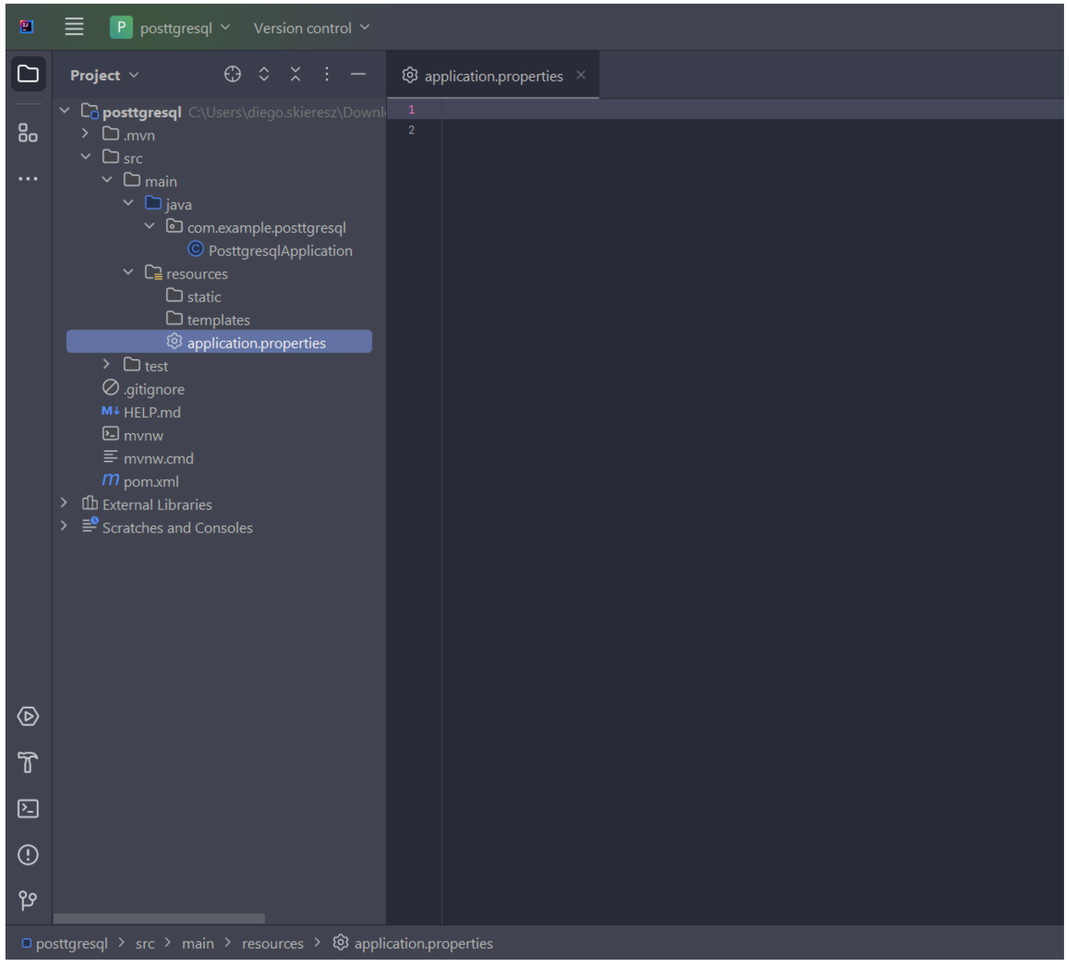


Estruturas de retrospectiva

Criar o database e editar o application.properties

- Neste aquivo especificamos diferentes propriedades do SpringBoot que queremos usar;

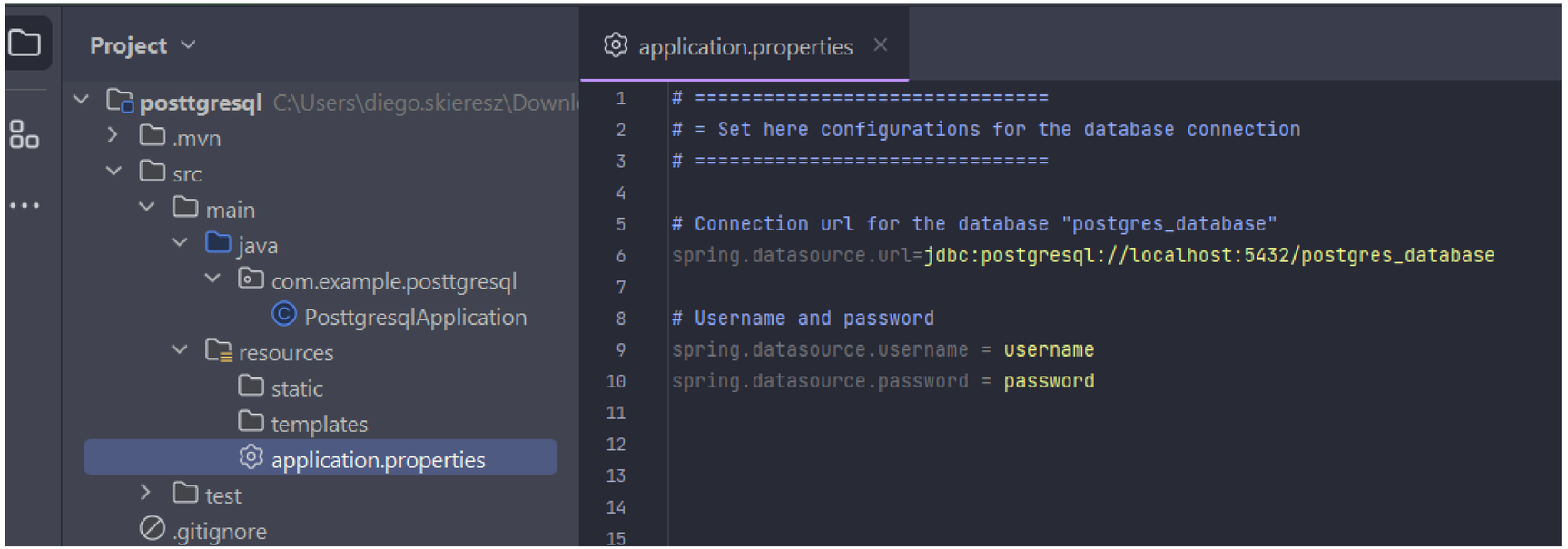
- Aqui onde iremos especificar a nossa string para conectar com o nosso database;



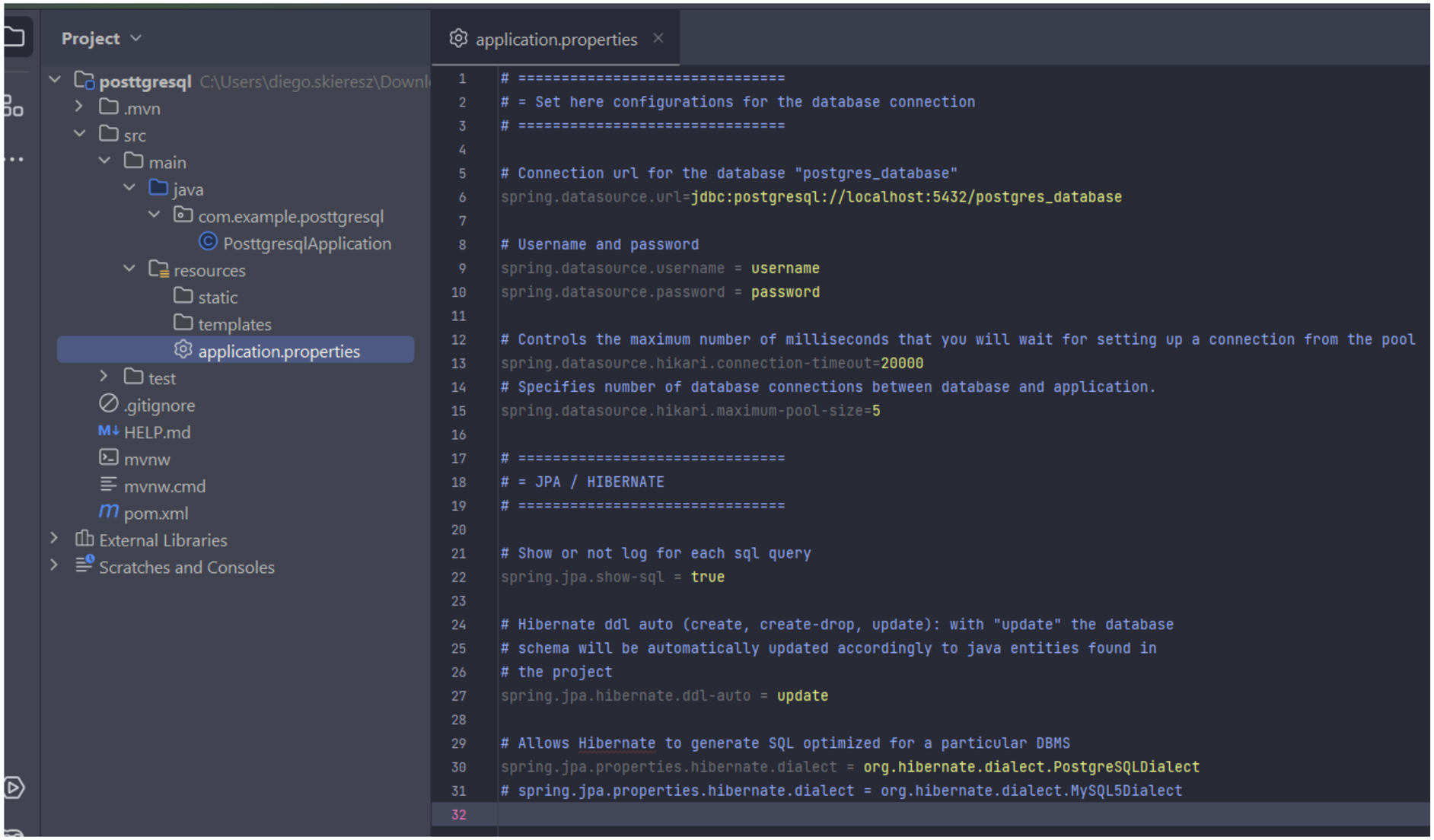
Configurando *application.properties*

Criar o database e editar o application.properties

- Primeiro definir a url de origem do banco, com a porta e o nome do database no final;
- Segundo, especificar o username;
- E também o **password** do “database”;



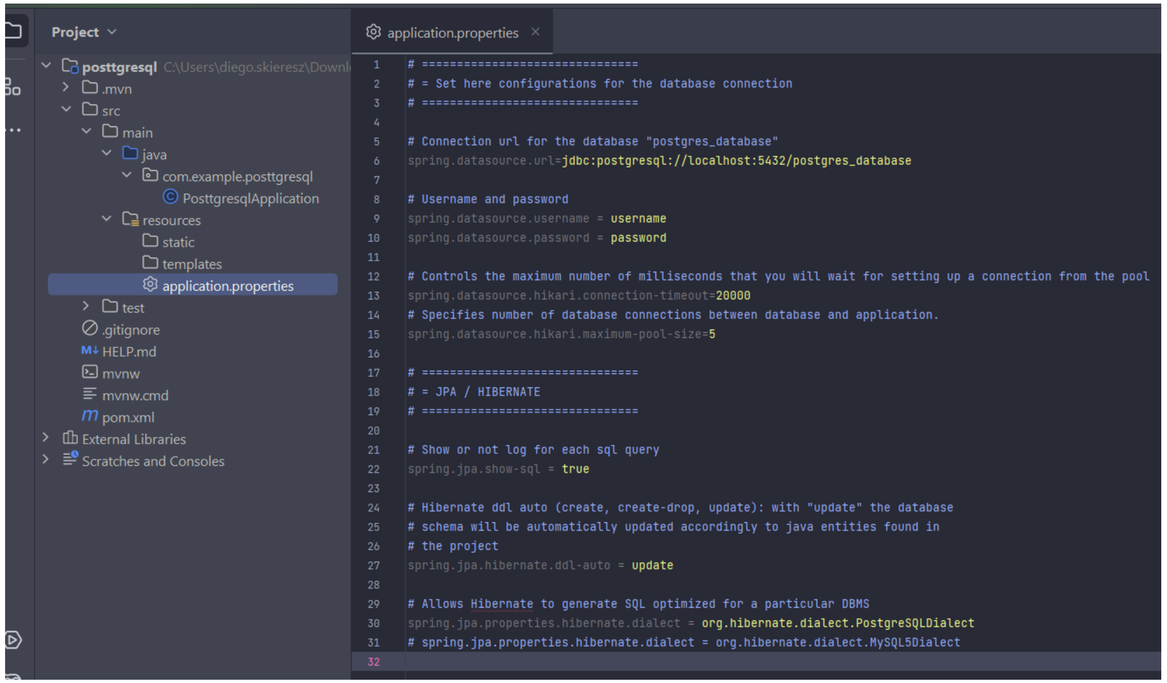
Configurações adicionais



Configurando *application.properties*

Criar o database e editar o application.properties

- **Connection timeout** = Quanto tempo para conexão deve disparar o timeout
- **Poolsize** = Máximo de conexões simultâneas ao database que queremos que o spring crie;
- **Dialect** = Qual dialeto do SQL iremos utilizar. O hibernate vai gerar queries automaticamente para o database;
- **ddl-auto** = Ele vai atualizar o database com a mudança, por exemplo se você muda o model e conecta ao database ele ira ver que existe uma mudança no model e vai executar um update ao database;
- **show-sql** = Loga o sql no console



Run the application

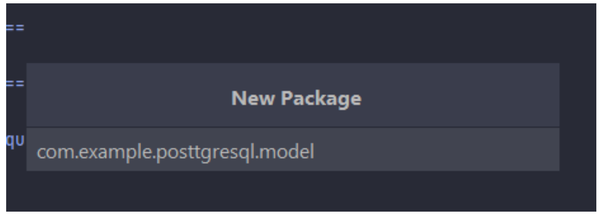
TIC em trilhas

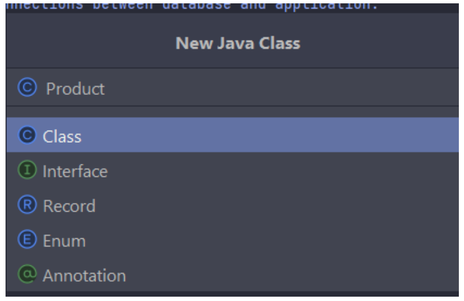
Model

- **O que é?** É como se fosse a "memória" da aplicação,
- onde guardamos informações importantes.
- Representa a camada de dados da aplicação.
- É responsável por gerenciar o estado e a lógica de negócios.
- No Spring, os objetos Model geralmente são POJOs (Plain Old Java Objects) que encapsulam os dados da aplicação.

- **Exemplo:** Se nossa aplicação é sobre uma lista de tarefas, o Model seria onde guardamos cada tarefa com suas informações, como nome, data, etc.

Criando o Model





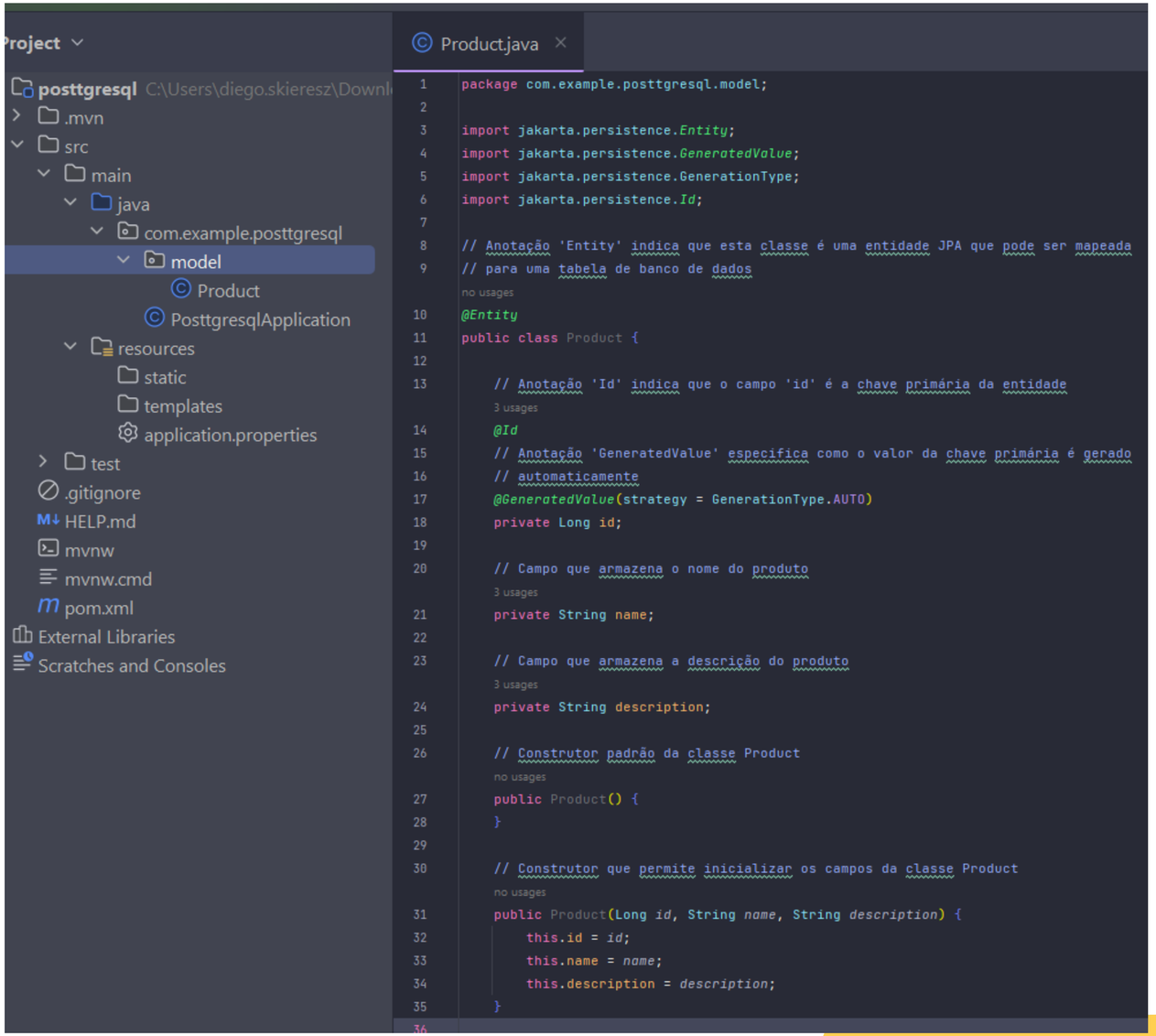
@Entity Diz ao Spring que é uma entidade no database

@Id é usada para marcar o campo id como a chave primária da entidade;

@GeneratedValue especifica como o valor da chave primária é gerado;

@GeneratedValue(strategy = GenerationType.AUTO) indica que a estratégia de geração da chave primária deve ser determinada automaticamente

- Usar no InteliJ o alt + insert para criar os construtores



Repository

- **O que é?** É como um assistente pessoal

para lidar com o armazenamento de informações.

Ele cuida de salvar e buscar coisas no "armário“

da aplicação (o banco de dados).

- É responsável pela interação com o banco de dados. Ele abstrai o acesso aos dados, permitindo que o Model se concentre na lógica de negócios.
- No Spring, os repositórios são frequentemente interfaces estendidas por interfaces do Spring Data JPA, que fornece métodos para realizar operações básicas de CRUD (Create, Read, Update, Delete) no banco de dados.

- **Exemplo:** Continuando com o exemplo, um repositório de tarefas poderia ter métodos como save(Task task), findById(Long id), etc.

Criando o Repository

Para criar um repositório e usar o model usaremos o **JpaRepository**

JpaRepository é uma interface fornecida pelo Spring Data JPA, possui funcionalidades predefinidas para realizar operações CRUD (Create, Read, Update e Delete) em entidades JPA.

Outras funcionalidades:

- **Métodos de Consulta Derivada**: O Spring Data JPA permite a criação de consultas personalizadas de forma declarativa, derivando o nome dos métodos na interface do repositório. Por exemplo, você pode criar métodos com nomes específicos, como findByFirstNameAndLastName, e o Spring Data JPA automaticamente gera a consulta SQL correspondente com base no nome do método..

Criando o Repository

Para criar um repositório e usar o model usaremos o **JpaRepository**

- **Paginação e Classificação**: O JpaRepository oferece suporte à paginação e classificação de resultados, facilitando a recuperação de grandes conjuntos de dados de forma eficiente.

- **Consultas Nativas e JPQL**: Além das consultas derivadas, o JpaRepository permite a execução de consultas SQL nativas e consultas JPQL (Java Persistence Query Language) personalizadas.

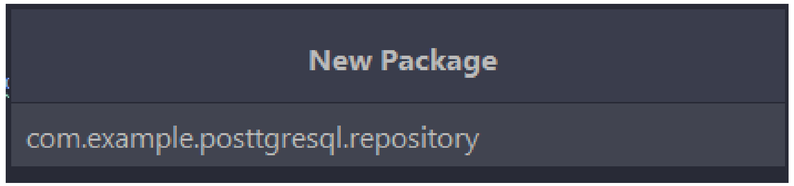
- **Métodos Customizados:** Você pode adicionar seus próprios métodos personalizados no repositório, para atender a requisitos específicos de consulta que não podem ser atendidos por consultas derivadas.

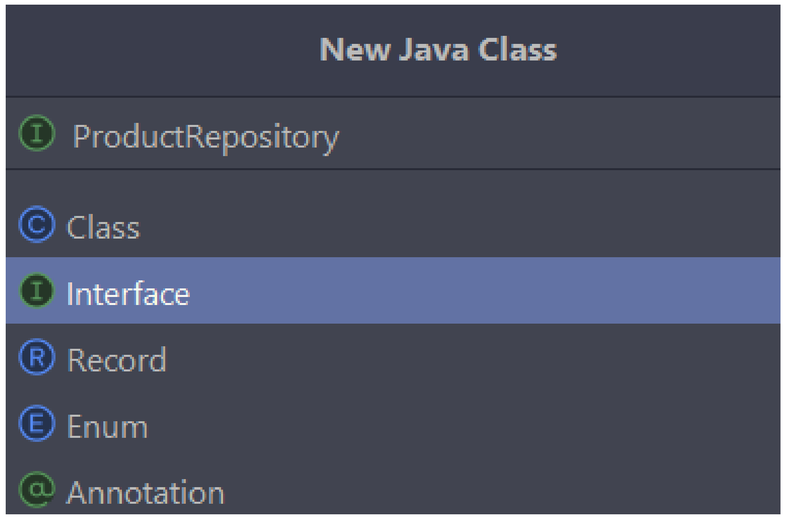
Criando o Repository

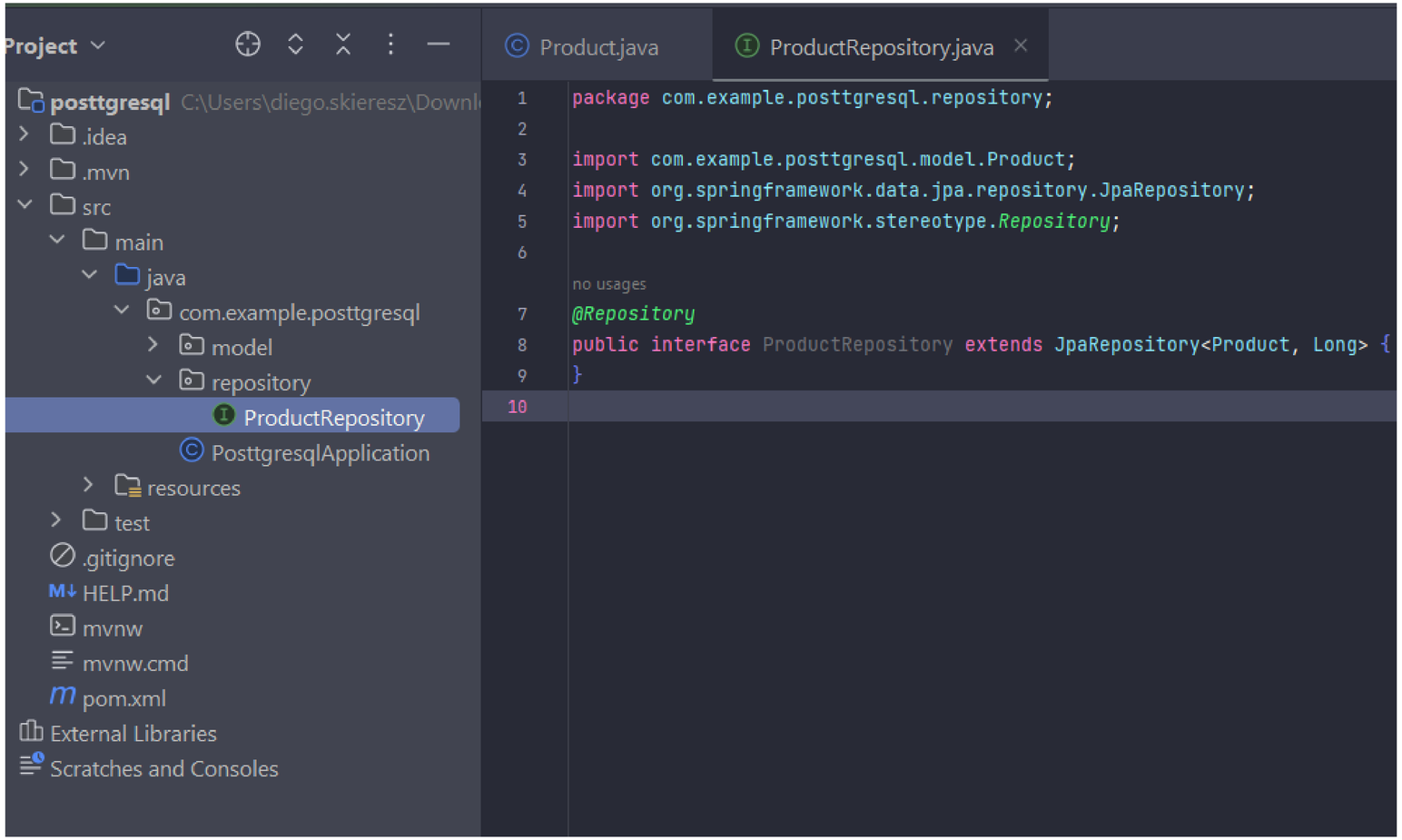
Existe outras maneiras de criar repositórios além do **JpaRepository**?

- **CrudRepository:** Esta é a interface mais básica e fornece métodos CRUD (Create, Read, Update e Delete) genéricos. É a interface pai do JpaRepository.
- **PagingAndSortingRepository:** Esta interface estende CrudRepository e adiciona suporte a paginação e classificação de resultados.
- **QuerydslPredicateExecutor:** Permite a execução de consultas complexas usando o Querydsl, que é uma biblioteca que fornece uma maneira tipada e segura de criar consultas dinâmicas.
- **KeyValueRepository**: Esta interface é usada quando você deseja trabalhar com armazenamento de chave-valor, em vez de um banco de dados relacional.
- **Repositórios de grafo**: O Spring Data também oferece suporte para repositórios orientados a gráficos, como Neo4jRepository, se você estiver usando um banco de dados de grafo como o Neo4j.

Criando o Repository







- Precisa configurar duas coisas, qual o tipo de dado tu esta armazenando no Repo (Product), e também qual o tipo do ID (Long)

Controller

- **O que é?** É o "cérebro" da aplicação, que recebe

as ordens (requisições) e decide o que fazer com elas.

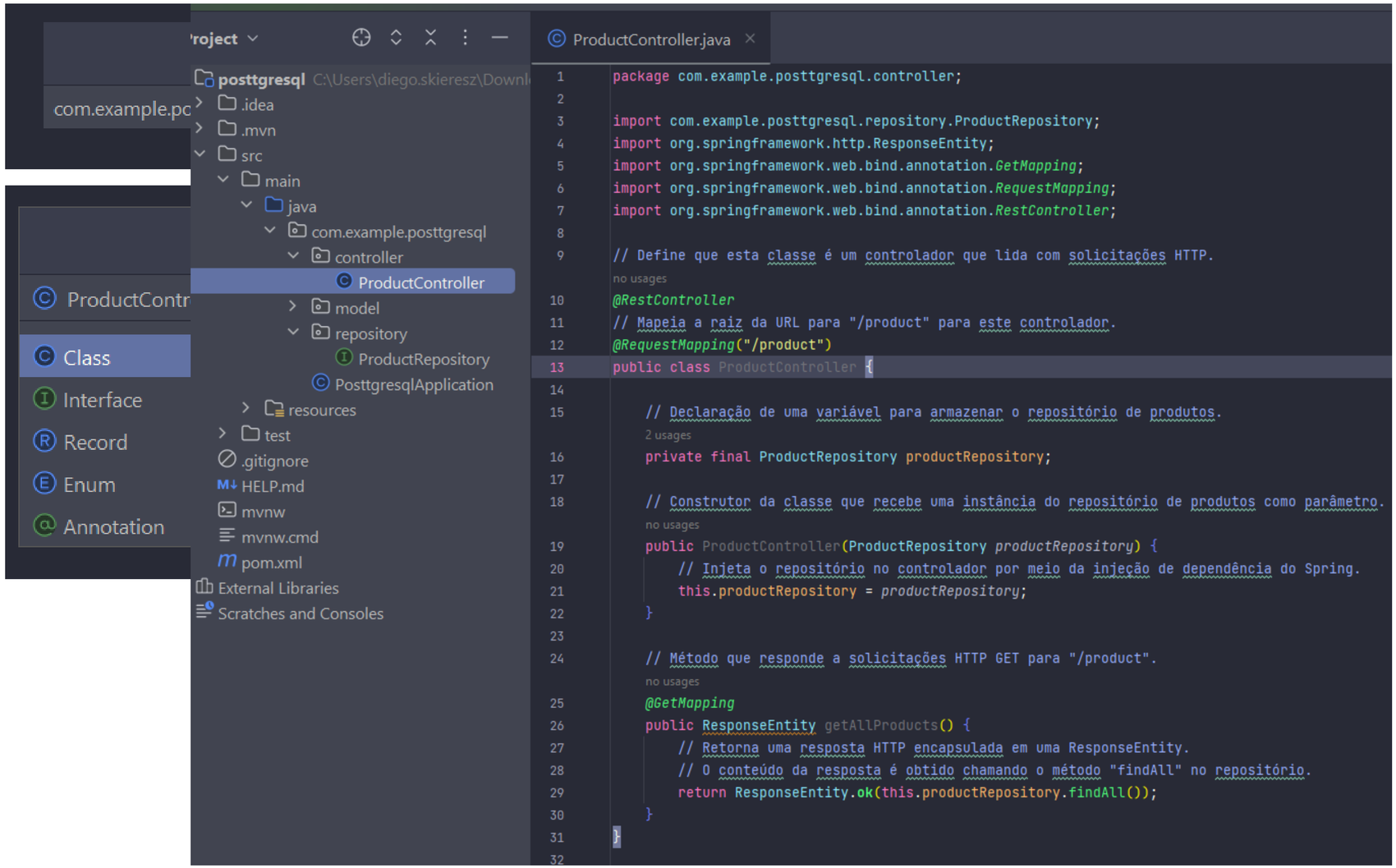
- Responsável por receber as requisições do cliente,

processar essas requisições e chamar os métodos apropriados no Model.

- No Spring, os controladores são geralmente classes anotadas com @Controller. Eles contêm métodos mapeados para URLs específicas.

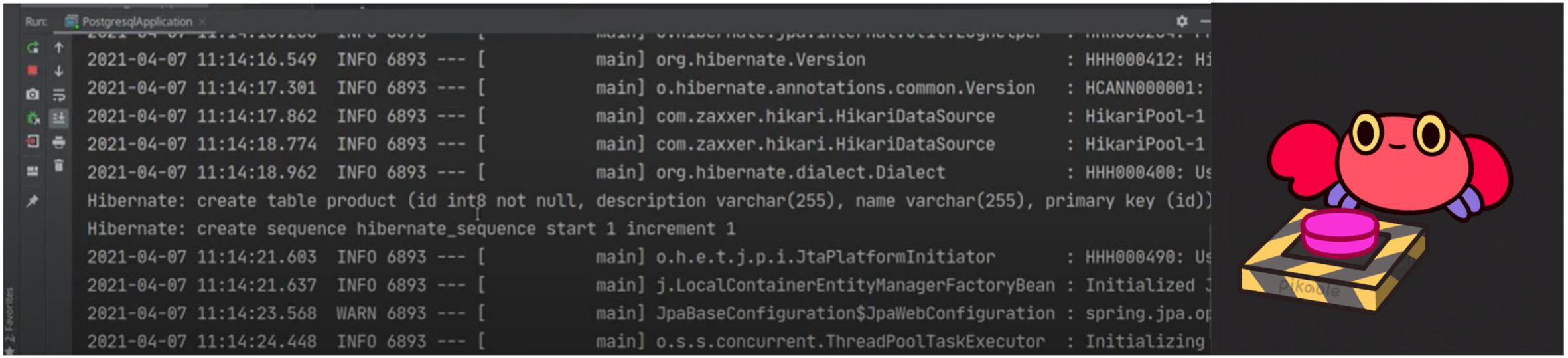
- **Exemplo:** Se alguém quer adicionar uma nova tarefa à lista, o Controller recebe essa ordem e avisa o Model para adicionar essa tarefa.

Criando o Controller



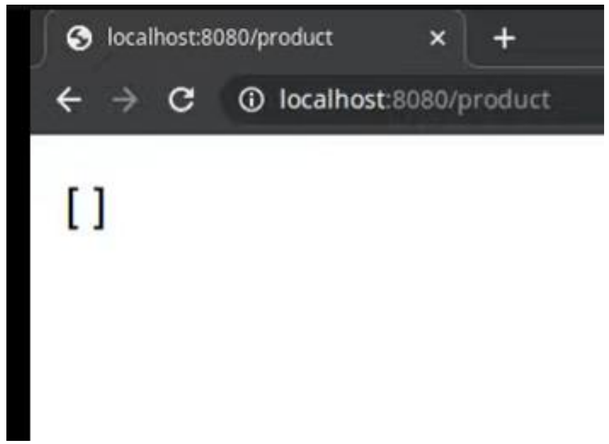
TIC em trilhas

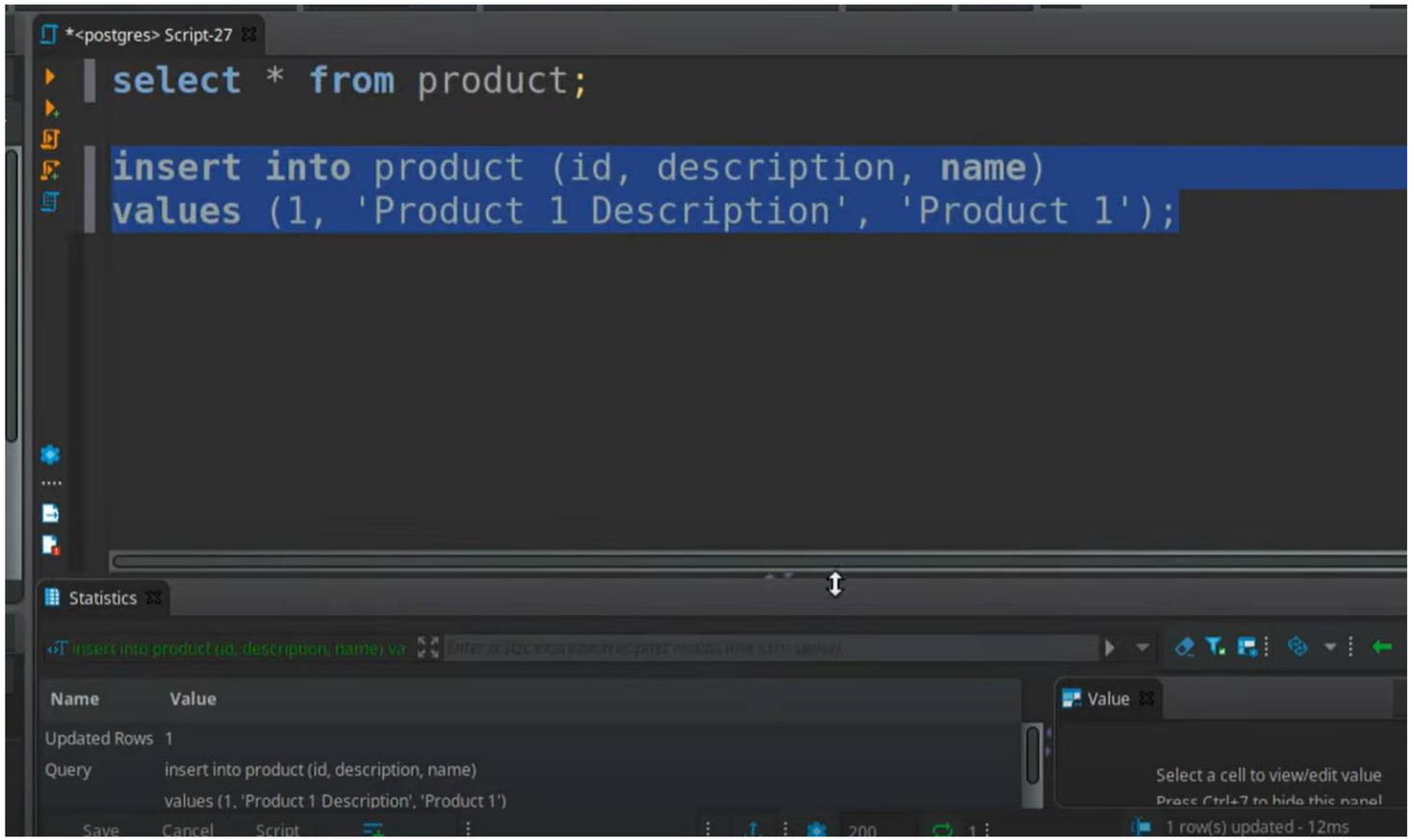
Start application



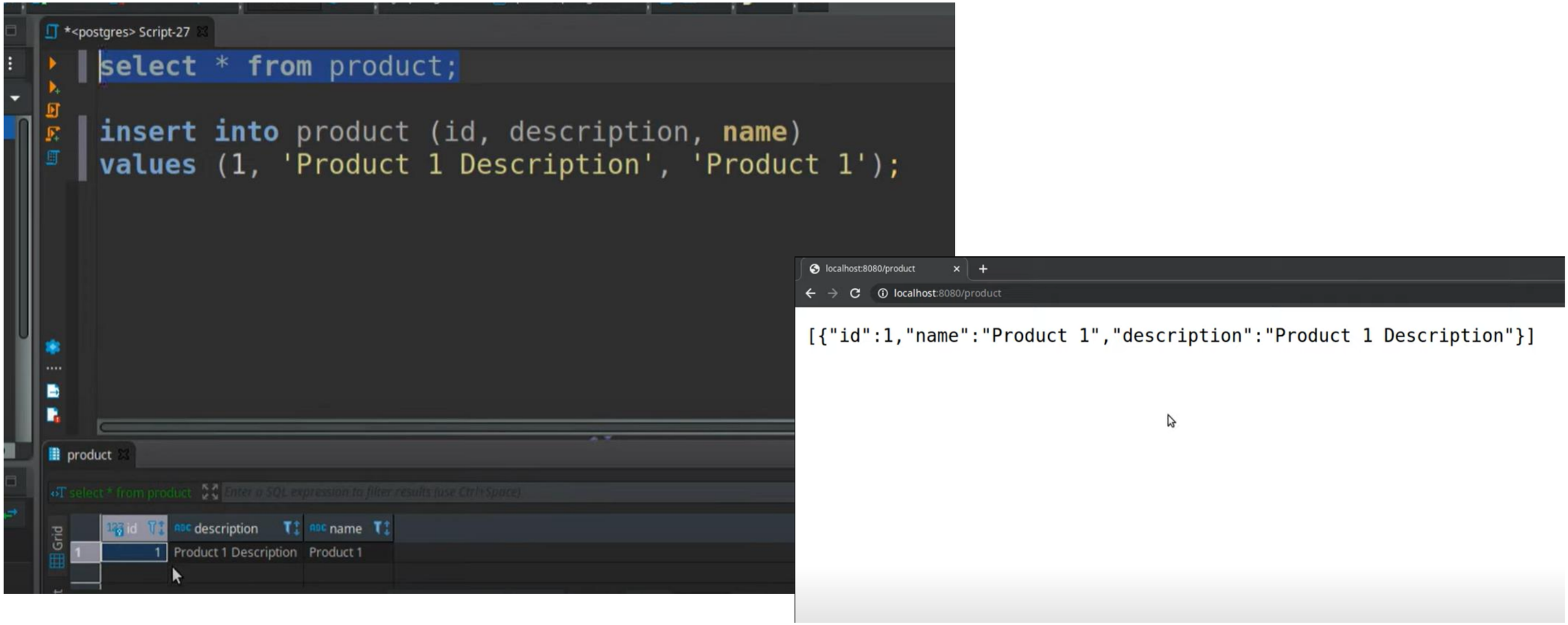
- Essa é a feature que utilizamos, **show-sql,** que mostra as queries que estamos executando no database;
- Como selecionamos o generatios type auto, ele esta criando a tabela "hibernate_sequence" onde, . Cada vez que uma nova entidade é persistida no banco de dados, o Hibernate consulta essa tabela para obter o próximo valor da sequência e o atribui à entidade antes de inseri-la no banco de dados.

Adding some data





Adding some data



Agradecemos a sua atenção!

TIC em trilhas