MÓDULO 15 – DOCKER – ATIVIDADE 05

OBJETIVO

Vamos rodar seu backend feito na aula 06 de typescript com um banco de dados Postgresql, cada um num container, ambos conectados a uma mesma rede do tipo bridge. Esse tipo de setup é comum em deploy de aplicações com Docker.

RESULTADOS

O passo zero foi pegar os arquivos referentes à aplicação da aula 06 de typescript e salvar na raiz de um novo projeto, onde o foco será a dockerização e o uso de rede tipo bridge para trafegar informações entre containers.

A seguir, vamos criar uma rede do tipo bridge, que é o tipo padrão. Portanto, se não especificamos o driver, será utilizado esse tipo de rede. Nesse tipo de rede, os containers ficam isolados e vão precisar usar a rede para se comunicarem entre si. Portanto, os containers ficam isolados de outros containers que não façam parte da rede criada. Quando é necessário utilizar outro tipo de rede, deve ser aplicada a opção "--driver", seguida do tipo que deseja-se atribuir a rede. Enfim, para criar a rede usamos esse comando:

docker network create --driver bridge net-aula05

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/reposi
torioHard/Alpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05_backtypescr
ipt$ docker network create --driver bridge net-aula05
593e8f6eb420ddc5605c826b1f85bd339d252d27e42fec7c7cb2a9546d961f44
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/reposi
```

Para mostrar que a rede foi criada com sucesso, podemos consultar a lista de redes, através desse comando:

docker network ls

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/reposi
torioHard/Alpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05_backtypescr
ipt$ docker network ls
NETWORK ID     NAME                                DRIVER    SCOPE
eb5c46411015   app_todoApp                         bridge    local
083ea36e9118   bridge                              bridge    local
a355973792d4   edtech-api-nodejs_default           bridge    local
f14782733f37   ex02_docker_and_prisma_default      bridge    local
6cc1bace930a   host                                host      local
593e8f6eb420   net-aula05                          bridge    local
```

Pode-se utilizar o comando "**docker network inspect net-aula05**" ou "**docker network inspect 593**" para obter informações detalhadas sobre a rede que foi criada. A imagem a seguir mostra algumas das informações obtidas. Note que existe uma chave "Driver" com o valor bridge, portanto, foi criado corretamente. Além disso, percebe-se uma chave "Gateway" com o valor 172.25.0.1. Esse valor indica o endereço IP da rede Docker do tipo bridge que foi criada. Trata-se do ponto de acesso que os containers devem utilizar para se comunicar com outras redes.

```
torioHard/Alpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05_backtypescr
ipt$ docker network inspect 593
[
    {
        "Name": "net-aula05",
        "Id": "593e8f6eb420ddc5605c826b1f85bd339d252d27e42fec7c7cb2a9546d96
1f44",
        "Created": "2023-05-21T09:24:39.905099942-03:00",
        "Scope": "local",
        "Driver": "bridge",
        "EnableIPv6": false,
        "IPAM": {
            "Driver": "default",
            "Options": {},
            "Config": [
                {
                    "Subnet": "172.25.0.0/16",
                    "Gateway": "172.25.0.1"
                }
            ]
```

O próximo passou foi criar um volume para persistir os dados do container que ficará responsável pelo database postgreSQL. Para criar um volume, utiliza-se esse comando:

docker volume create vol-aula05

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/reposi
torioHard/Alpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05_backtypescr
ipt$ docker volume create vol-aula05
vol-aula05
```

Através das informações que estão disponíveis no link https://hub.docker.com/_/postgres/ , aplicou-se o comando a seguir:

docker run -d --name container-db-aula05 --network net-aula05 --network-alias alias-rede-postgresql -v vol-aula05:/var/lib/postgresql/data -e POSTGRES_USER=letsadmin -e POSTGRES_PASSWORD=1234 -e POSTGRES_DB=db-aula05 postgres

Explicando o comando acima: **docker run -d** cria e executa um container em segundo plano; a **opção "--name"** indica o nome atribuído a esse container; a **opção "--network"** indica o nome da rede que o container fará parte; a **opção "--network-alias"** indica um apelido para o container do banco de dados na rede, que é útil para quando algum outro container deseje fazer referência ao container de banco de dados; a **flag "-v"** indica o volume e o caminho onde os dados persistem; a **flag "-e"** indica variáveis de ambiente, sendo a variável POSTGRES_PASSWORD obrigatória, pois define a senha de superusário, portanto, é necessário informá-la. Mesmo não sendo obrigatório, optou-se por adicionar duas outras variáveis: uma para o nome do database (POSTGRES_DB) e

outra para o superusuário (POSTGRES_USER); por fim, **postgres** indica a imagem que foi utilizada como base na criação do container.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/reposi
torioHard/Alpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05_backtypescr
ipt$ docker run -d --name db-aula05 --network net-aula05 --network-alias al
ias-rede-postgresql -v vol-aula05:/var/lib/postgresql/data -e POSTGRES_USER
=letsadmin -e POSTGRES_PASSWORD=1234 -e POSTGRES_DB=db-aula05 postgres
783e600d9b998547c0df6646a9e05809ae7c82ed83359f2e8ca6b893b77fc24c
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/reposi
```

Para confirmar que o container foi criado com sucesso, aplicou-se o comando a seguir:

docker ps

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01
ktypescript$ docker ps
CONTAINER ID   IMAGE      COMMAND                  CREATED          STATUS          PORTS      NAMES
783e600d9b99   postgres   "docker-entrypoint.s…"   16 minutes ago   Up 16 minutes   5432/tcp   db-aula05
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01
```

Para obter informações detalhadas e confirmar que o container está conectado a rede bridge “net-aula05”, pode-se aplicar esse comando:

docker inspect 783

Como resultado do comando anterior, um JSON com informações detalhadas sobre o container aparecem no terminal. Na imagem abaixo, percebe-se que o container está conectado à rede que criamos no início dessa resolução (net-aula05). A chave Aliases possui apelidos que podem ser empregados para se referir a esse container. Além disso, a chave “Gateway” indica o endereço IP que deve ser usado pelos containers na rede como ponto de acessos para se comunicarem com outras redes ou com o mundo externo. Por fim, a chave “IPAddress” representa o endereço IP atribuído ao container dentro da rede "net-aula05", o que permite a comunicação interna entre containers na mesma rede com o container “db-aula05”. Portanto, quando outro container for fazer acesso e necessite passar as credenciais, uma informação necessária é o host. Nesse caso, utilizaremos o endereço IP do container postgres nessa rede que foi criada no início.


```
        "MacAddress": "",
        "Networks": {
            "net-aula05": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": [
                    "alias-rede-postgresql",
                    "783e600d9b99"
                ],
                "NetworkID": "593e8f6eb420ddc5605c826b1f85bd339d252d27e42fec7c7cb2a9546d961f44",
                "EndpointID": "00c16672ad855cecd7a39ff5c325cd9d74d5e1541fc76470f1d1c2bb936c5c74",
                "Gateway": "172.25.0.1",
                "IPAddress": "172.25.0.2",
                "IPPrefixLen": 16,
```


Portanto, apenas para dar destaque o endereço IP do container que armazena o bando de dados é 172.25.0.2 .

A próxima etapa é acessar esse banco de dados e criar o schema do nosso database, isto é, as tabelas que fazem parte da nossa aplicação. A seguir, vamos recordar o que deve ter no database.

O database é composto por uma tabela chamada accounts. Caso estivessemos no PSQL ou no PGADMIN, poderíamos criá-la através dessa query:

```
CREATE TABLE accounts (
   id UUID PRIMARY KEY DEFAULT  uuid_generate_v4(),
   name TEXT NOT NULL UNIQUE,
   email TEXT NOT NULL UNIQUE,
   password TEXT NOT NULL,
   created_at TIMESTAMP DEFAULT NOW(),
   updated_at TIMESTAMP,
   deleted_at TIMESTAMP,
   deleted BOOLEAN DEFAULT FALSE
);
```

Para adicionar elementos a essa tabela, podemos utilizar o comando a seguir:

```
INSERT INTO accounts (
   name,
   email,
   password
)
VALUES
('lets','lets@gmail.com','123'),
('maria','maria@gmail.com','222'),
('joao','joao@gmail.com','333'),
('felipe','felipe@gmail.com','444');
SELECT * FROM accounts;
```

Levando em consideração o Docker, devemos aplicar o seguinte comando para acessar o database db-aula05 que está no container:

docker exec -it db-aula05 psql -U letsadmin -d db-aula05

Detalhando o comando acima: **docker exec** permite usar um comando dentro de um container ativo; a **flag "-it"** habilta a interação via terminal, permitindo a inserção de comandos e visualização da saída; o **primeiro "db-aula05"** é o nome do container (poderia ter sido usado o ID do container, por exemplo) que desejamos conectar; **psql** é um cliente que permite interagir com bancos de dados que são geridos pelo sistema de gerenciamento de banco de dados relacional PostgreSQL; a **flag "-u"** especifica o nome do usuário usado na conexão, nesse caso optou-se pelo superusuário definido anteriormente; a **flag "-d"** aponta o nome do banco de dados que desejamos conectar e que está dentro do container, que nesse caso coincide com o nome do container, mas não é uma regra.

Na imagem a seguir, mostra-se que estamos conectados ao banco de dados "db-aula05".

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repo
ktypescript$ docker exec -it db-aula05 psql -U letsadmin -d db-aula05
psql (15.2 (Debian 15.2-1.pgdg110+1))
Type "help" for help.

db-aula05=#
```

No schema da tablea accounts, tem uma chave primária do tipo UUID. Optou-se por adicionar um valor padrão através da função uuid_generate_v4() que adiciona um valor válido de UUID caso não tenha sido informado algum valor para esse campo na hora da adição de um novo elemento. No entanto, antes de criar a tabela, precisamos adicionar uma extensão que permitirá o uso dessa função. No terminal do psql utiliza-se esse comando:

CREATE EXTENSION IF NOT EXISTS "uuid-ossp";

```
db-aula05=# CREATE EXTENSION IF NOT EXISTS "uuid-ossp";
CREATE EXTENSION
db-aula05=#
```

A próxima etapa é copiar e colar os comandos definidos anteriormente, que contém o nome da tabela do nosso banco de dados e a adição de elementos.

```
CREATE TABLE accounts (
    id UUID PRIMARY KEY DEFAULT  uuid_generate_v4(),
    name TEXT NOT NULL UNIQUE,
    email TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP,
    deleted_at TIMESTAMP,
    deleted BOOLEAN DEFAULT FALSE
);
INSERT INTO accounts (
    name,
    email,
    password
)
VALUES
('lets','lets@gmail.com','123'),
('maria','maria@gmail.com','222'),
```

('joao','joao@gmail.com','333'),

('felipe','felipe@gmail.com','444');

SELECT * FROM accounts;

As próximas imagens mostram o resultado obtido no terminal do psql:

```
db-aula05=# CREATE TABLE accounts (
    id UUID PRIMARY KEY DEFAULT  uuid_generate_v4(),
    name TEXT NOT NULL UNIQUE,
    email TEXT NOT NULL UNIQUE,
    password TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP,
    deleted_at TIMESTAMP,
    deleted BOOLEAN DEFAULT FALSE
);
CREATE TABLE
db-aula05=#
```

```
db-aula05=# INSERT INTO accounts (
    name,
    email,
    password
)
VALUES
('lets','lets@gmail.com','123'),
('maria','maria@gmail.com','222'),
('joao','joao@gmail.com','333'),
('felipe','felipe@gmail.com','444');
INSERT 0 4
db-aula05=#
```

Através do comando "**SELECT * FROM accounts;**" é possível visualizar uma tabela com todas as informações da tabela accounts.

```
db-aula05=# SELECT * FROM accounts;
                  id                  |  name  |      email       | password |         created_at         |
--------------------------------------+--------+------------------+----------+----------------------------+-
 ee81ec65-5dfa-4a6d-909d-878efe26bd33 | lets   | lets@gmail.com   | 123      | 2023-05-21 16:16:27.299123 |
 96128a95-21c9-4fb6-a1fd-70b76c36d696 | maria  | maria@gmail.com  | 222      | 2023-05-21 16:16:27.299123 |
 9efab61c-526f-45fa-91ca-36051b044520 | joao   | joao@gmail.com   | 333      | 2023-05-21 16:16:27.299123 |
 6ac1ad56-2d4f-4817-bf6a-1f84e0df5044 | felipe | felipe@gmail.com | 444      | 2023-05-21 16:16:27.299123 |
(4 rows)

db-aula05=#
```

Caso necessário limpar o terminal do psql, basta utilizar "\! clear". Para encerrar a conexão via psql, podemos usar "\q"

```
db-aula05=# \q
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Mo
ktypescript$
```

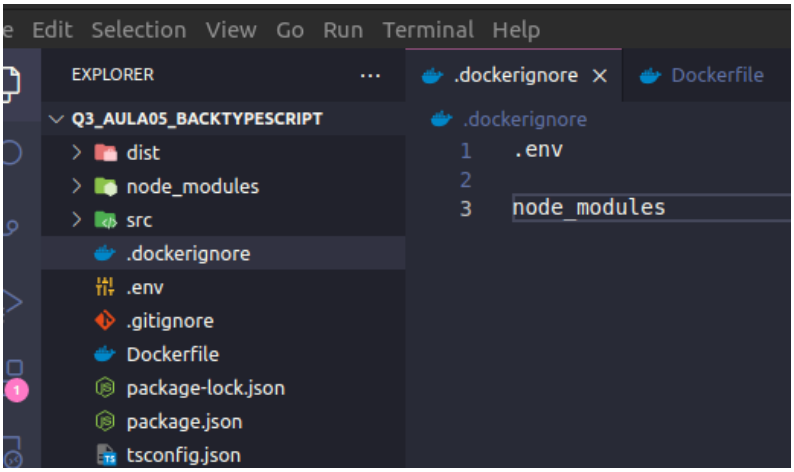
A próxima etapa é criar um arquivo Dockerfile na raiz do projeto de backend. A seguir, apresenta-se o conteúdo do Dockerfile:

syntax=docker/dockerfile:1

```
FROM node:18-alpine

WORKDIR /appbackend

COPY . .

RUN npm install

CMD ["node", "dist/server.js"]

EXPOSE 80
```

Explicando o que o Dockerfile faz: Foi definida a imagem base da nova imagem que vamos criar como node versão 18-alpine. Define-se um diretório de trabalho como appbackend; Todos os arquivos do diretório atual será copiado para dentro do container (exceto o que será definido no arquivo .dockerignore, a saber, arquivo .env e node_modules/). Em seguida, executa-se o comando npm install, que instalará todas as dependências especificadas no arquivo package.json dentro do container. Quando um container é executado a partir dessa imagem, o comando padrão é “node dist/server.js” que iniciará o servidor em modo de produção; A porta exposta dentro do container quando for criado será a porta 80.

A próxima etapa é criar um arquivo .dockeignore e inserir dentro dele .env e node_modules/, para que sejam ignorados e não seja copiados para dentro do container. A imagem a seguir mostra o conteúdo do .dockerignore e a organização do projeto.



A próxima etapa foi criar uma imagem a partir do Dockerfile, com o terminal na raiz do projeto (mesmo local do Dockerfile). Para criar a imagem, utilizamos o comando a seguir:

docker build -t image-backend-typescript .

No comando acima, build indica que desejamos criar uma imagem, atribuindo uma tag a essa imagem, que deve ser construída a partir de um arquivo Dockerfile presente no diretório atual do terminal (.).

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/
repositorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05
_backtypescript$ docker build -t image-backend-typescript .
[+] Building 7.3s (12/13)
 => [internal] load build definition from Dockerfile                0.6s
 => => transferring dockerfile: 174B                                0.1s
 => [internal] load .dockerignore                                   0.7s
```

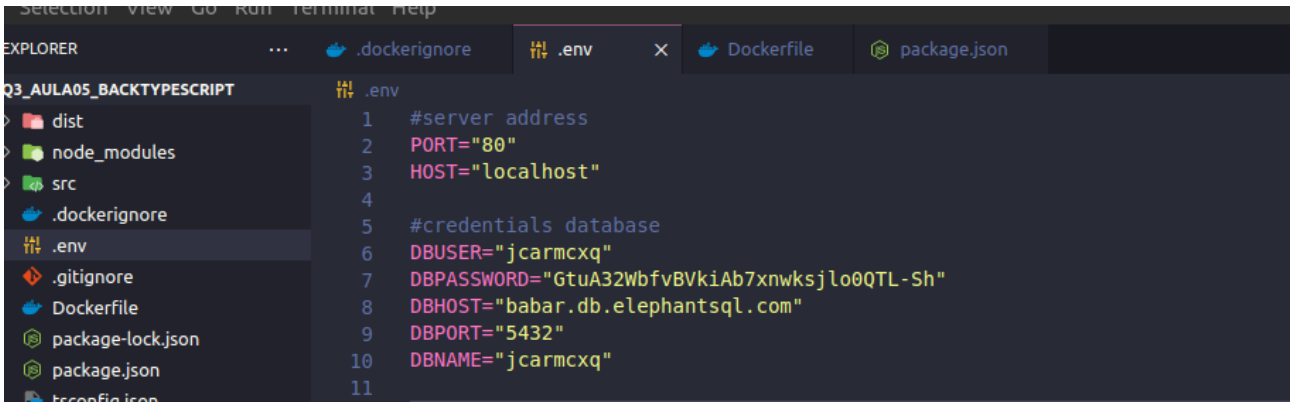
```
 => => writing image sha256:e96c1569501f64398f555a7d6b38036144  0.1s
 => => naming to docker.io/library/image-backend-typescript     0.1s
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/
```

Para verificar que a imagem foi criada com sucesso e está disponível, podemos usar esse comando:

docker image ls

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cyc
ktypescript$ docker image ls
REPOSITORY                    TAG       IMAGE ID       CREATED              SIZE
image-backend-typescript      latest    e96c1569501f   About a minute ago   235MB
image-todo-app                latest    0ea4022eb3d2   3 days ago           265MB
lets2/first-image             latest    0f3fff89387b   5 days ago           177MB
first-image                   latest    0f3fff89387b   5 days ago           177MB
```

A próxima etapa é criar e executar um container a partir dessa imagem. Durante a criação do container, será necessário definir algumas variáveis de ambiente. O nome delas precisam ser iguais ao conteúdo do arquivo .env, pois o backend precisa delas para funcionar. A imagem a seguir mostra as variáveis que precisam ser definidas:



Note que essas credenciais não são as do container do banco de dados que foi criada mais cedo, é necessário fazer uma adaptação para que corresponda. Uma informação importante é que em DBHOST, podemos utilizar o apelido (alias) atribuído ao container na opção “--network-alias”, a saber, **alias-rede-postgresql** . Caso não tivéssemos atribuído um apelido para o container, teríamos que utilizar o IP dele na rede (valor presente na chave IPAddress, visualizado com o comando docker inspect). A saber, o IP do container na rede interna bridge é 172.25.0.2 . Portanto, as variáveis de ambiente que devem ser levadas em consideração e seus respectivos valores são:

PORT=80

HOST=0.0.0.0

DBUSER=letsadmin

DBPASSWORD=1234

DBHOST=alias-rede-postgresql

DBPORT=5432

DBNAME=db-aula05

Observação: Na imagem mostrada anteriormente, usava-se localhost, no entanto, seguindo o processo de executar os containers, apareceu um problema onde dentro do container do backend, de modo que ao mapear a rota e entar acessar pelo host o localhost:80/accounts, aparecia uma mensagem de que a página foi recarregada ou mensagem de erro, quando deveria na verdade mostrar um json com as 4 contas adicionadas. Enfim, após pesquisar uma solução que funcionou foi trocar de localhost para 0.0.0.0 (ver https://stackoverflow.com/questions/64666842/docker-run-connection-was-reset-while-the-page-was-loading). Uma explicação dada pelo chatgpt quando questionei a diferença entre usar localhost e 0.0.0.0 foi:

> "Ao definir o valor de HOST como localhost, o servidor do backend dentro do contêiner estará vinculado apenas à interface de loopback do contêiner, o que significa que ele só será acessível dentro do próprio contêiner. Essa configuração é adequada quando você deseja que o backend seja acessível somente internamente.
>
> Por outro lado, ao definir o valor de HOST como 0.0.0.0, o servidor do backend estará vinculado a todas as interfaces disponíveis no contêiner. Isso permite que o backend seja acessado de outros contêineres ou da rede externa, desde que as portas estejam corretamente mapeadas e as configurações de rede estejam adequadas.
>
> Portanto, se você deseja que o backend seja acessível de outros contêineres ou da rede externa, é recomendável definir o valor de HOST como 0.0.0.0 para permitir conexões de entrada de qualquer interface. No entanto, se o acesso for restrito apenas ao próprio contêiner, você pode manter o valor de HOST como localhost."

A seguir, apresenta-se o comando para criar o container a partir da imagem da aplicação de backend em typescript, considerando as variáveis de ambiente mostradas anteriormente, de forma que o novo container faça parte da mesma rede (net-aula05), mapeando a porta 80 do container com a 80 do host (máquina local) e lembrando que na imagem tinha um comando padrão para quando o container é executado (node dist/server.js).

docker run -dp 80:80 --name container-backend-ts --network net-aula05 --network-alias alias-rede-backend -e PORT=80 -e HOST=0.0.0.0 -e DBUSER=letsadmin -e DBPASSWORD=1234 -e DBHOST=alias-rede-postgresql -e DBPORT=5432 -e DBNAME=db-aula05 image-backend-typescript

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05_bac
ktypescript$ docker run -dp 80:80 --name container-backend-ts --network net-aula05 --network-alias alias-rede-backend -e  PORT=80 -e HOST=0.0.
0.0 -e DBUSER=letsadmin -e DBPASSWORD=1234 -e  DBHOST=alias-rede-postgresql -e  DBPORT=5432 -e DBNAME=db-aula05 image-backend-typescript
8a87f8eeff7a0d1fbd2b221748ca4a2d1855817fae182f103a7a696f5211c87b
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05_bac
ktypescript$
```

Para confirmar que o container foi criado corretamente e está em executação, podemos usar o comando “docker ps”, conforme ilustrado a seguir:

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module15-Doc
ktypescript$ docker ps
CONTAINER ID   IMAGE                      COMMAND                  CREATED         STATUS         PORTS
8a87f8eeff7a   image-backend-typescript   "docker-entrypoint.s…"   3 minutes ago   Up 3 minutes   0.0.0.0:80->80/tcp,
r-backend-ts
783e600d9b99   postgres                   "docker-entrypoint.s…"   7 hours ago     Up 7 hours     5432/tcp
5
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module15-Doc
```

Para obter informações detalhadas sobre o container, pode-se empregar o comando a seguir:

docker inspect 8a87


```
        "MacAddress": "",
        "Networks": {
            "net-aula05": {
                "IPAMConfig": null,
                "Links": null,
                "Aliases": [
                    "alias-rede-backend",
                    "8a87f8eeff7a"
                ],
                "NetworkID": "593e8f6eb420ddc5605c826b1f85bd339d252d27e42fec7c7cb2a9546d961f44",
                "EndpointID": "0eea5e0945e3e8ef2c124f22ea03f86849c7835976d79f2d27e6404653d1d671",
                "Gateway": "172.25.0.1",
                "IPAddress": "172.25.0.3",
                "IPPrefixLen": 16,
                "IPv6Gateway": "",
                "GlobalIPv6Address": "",
                "GlobalIPv6PrefixLen": 0,
                "MacAddress": "02:42:ac:19:00:03",
                "DriverOpts": null
            }
        }
```


Nota-se que o container está conectado a rede “net-aula05”, o que permitirá a comunicação entre os containers que fazem parte dessa rede. Outras duas informações relevantes são: “Gateway”:172.25.0.1, que indica o endereço IP de que deve ser utilizado para que ocorra o acesso de outras redes ou rede externa com os containers que fazem parte dessa rede; e “IPAddress”:172.25.0.3, que indica o endereço IP do container backend que foi recém-criado. Esse é o endereço que outros containers devem utilizar para se comunicar com o backend. Por fim, nota-se que também possui um alias (apelido), a saber “alias-rede-backend”.

Vamos acessar o container backend para verificar que o arquivo .env não foi transferido para lá. Para tal, vamos abrir um terminal, usando

docker exec -it container-backend-ts sh

Esse comando permite abrir um terminal sh dentro do container. Utilizando-se “**ls -a**”, nota-se uma lista dos arquivos e pastas dentro do diretório, ocultos inclusive. Não existe nenhum arquivo “.env”, confirmando que não foi transferido (estava presente o nome no arquivo .dockerignore). O node_modules está presente porque ficou definido no Dockerfile um comando **npm install** para ser executado.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Modul
ktypescript$ docker exec -it container-backend-ts sh
/appback # ls -a
.                .dockerignore    Dockerfile       node_modules     package.json     tsconfig.json
..               .gitignore       dist             package-lock.json src
/appback #
```

Ao tentar fazer uma conexão usando o pgadmin não funcionou. Para funcionar, em vez de informar o network-alias do container do postgres, usei o ip dele na rede (172.25.0.2). A saber todas as credenciais que precisei no pgadmin foram:

DBUSER=letsadmin

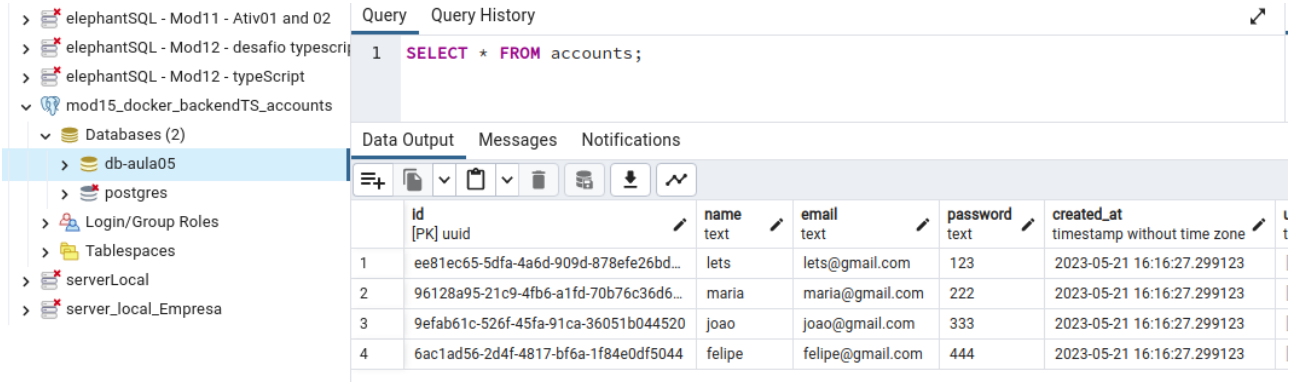
DBPASSWORD=1234

DBHOST=172.25.0.2

DBPORT=5432

DBNAME=db-aula05

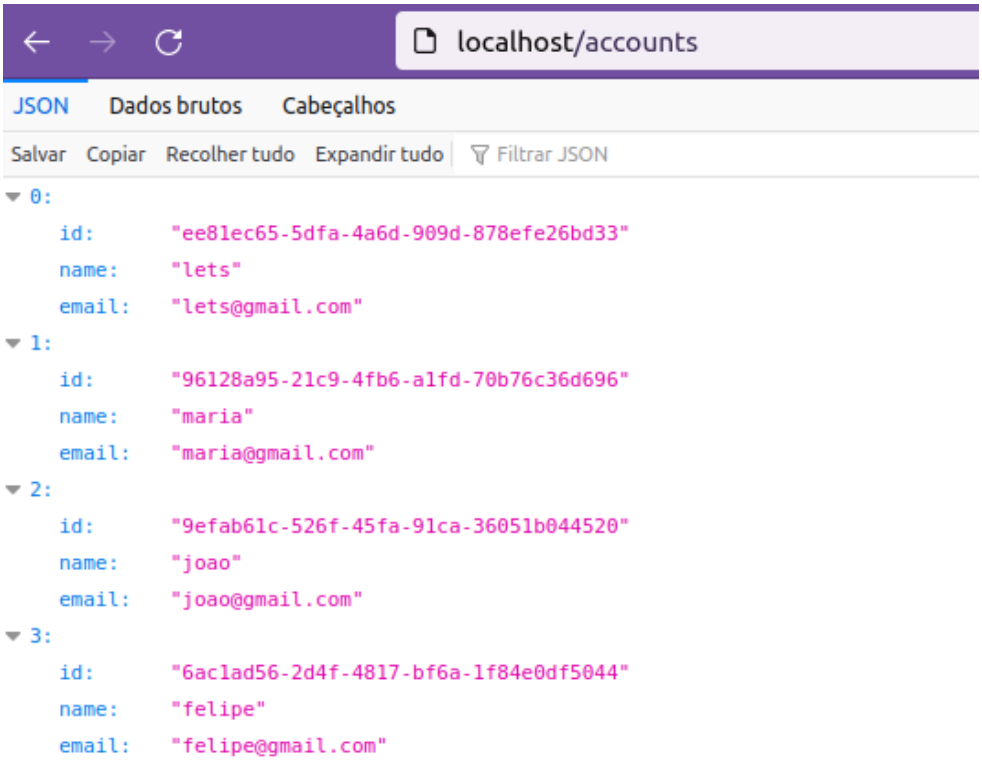
Na figura abaixo, observa-se o resultado de um comando de seleção, onde obtemos as informações da tabela accounts, via pgadmin:



O comando “**docker logs -f 8a87**” exibe os logs de um container em tempo real.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositori
ktypescript$ docker logs -f 8a87
Server is running on https://0.0.0.0:80
```

Indo no navegador, acessando http://localhost:80/accounts (não é necessário o 80 porque é a porta padrão do http, mas deixei para ficar explícito)

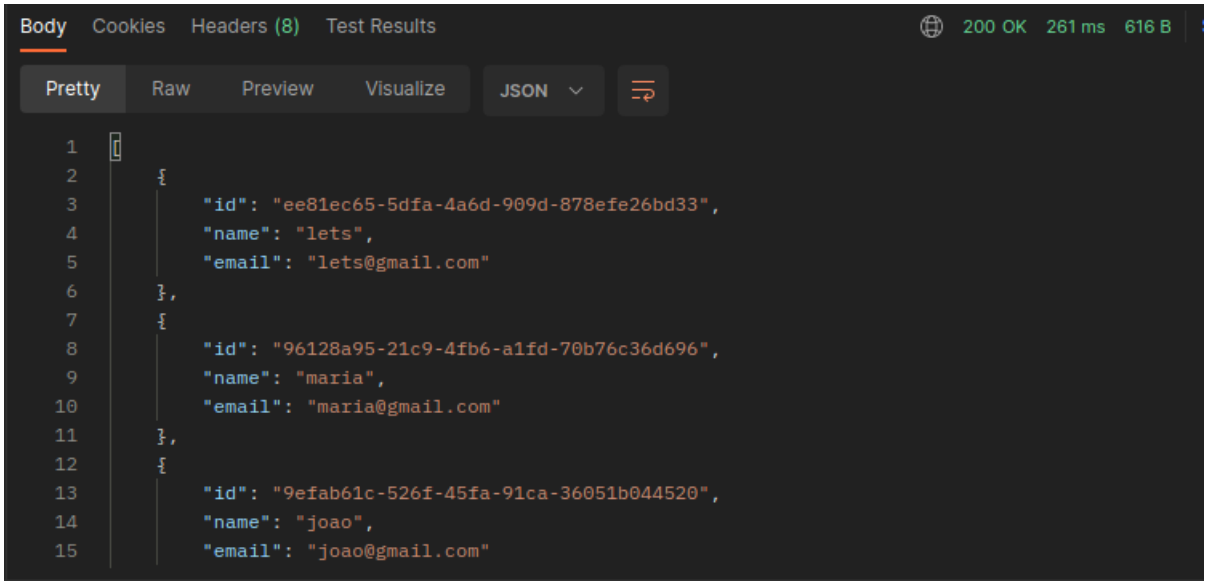


Nota-se que funcionou corretamente, recebemos as informações que foram adicionadas via psql em outro momento. Um ponto que quero destacar é o seguinte: no pgadmin não consegui usar o network-alias para acessar o container do postgres, foi necessário usar o ip do container (172.25.0.2), porém o container do backend que está na mesma rede do postgres (net-aula05) conseguiu acessar passando como variável de ambiente DBHOST=alias-rede-postgresql . A

A última etapa foi empregar o Postman para utilizar as rotas e verificar se realmente está alterando as informações do container.

Caso 1: GET (http://localhost:80/accounts) - Visualizar as informações de todas as contas registradas.

Response:



Caso 2: POST (http://localhost:80/accounts/login) - Faz o login a partir de uma conta já registrada, pois algumas rotas exigem autenticação.

Body:

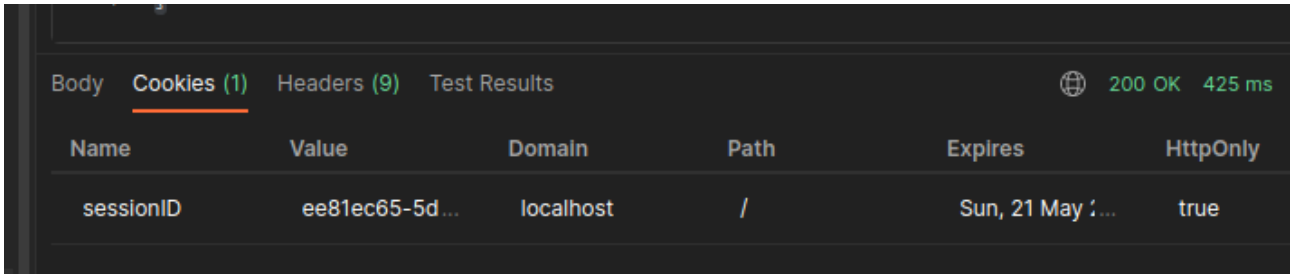
{

"email":"lets@gmail.com",

"password":"123"

}

Response:



Caso 3: POST (http://localhost:80/accounts) – Registrar uma nova container

Body:

{

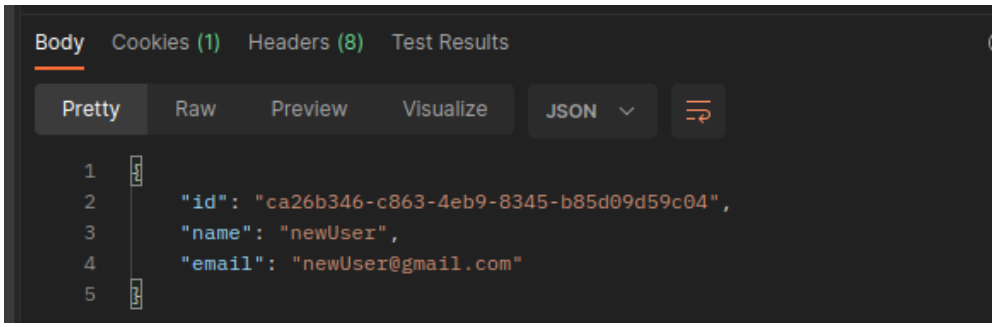
"name":"newUser",

"email":"newUser@gmail.com",

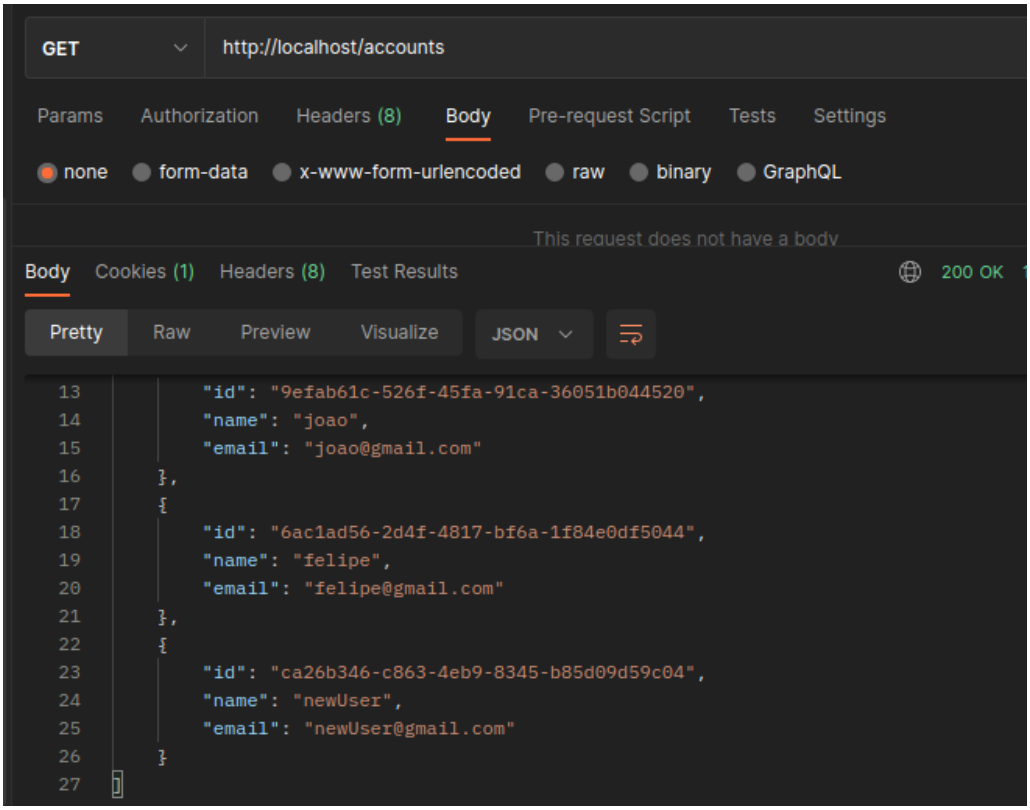
"password":"555"

}

Response:



Para confirmar que foi adicionado com sucesso no banco de dados, vamos usar novamente a rota que pega todas as contas:



Nota-se que o último elemento é o usuário que acabou de ser criado.

Caso 4: PATCH (http://localhost:80/accounts/) - Atualiza as informações de um usuário. Apenas o usuário que está logado pode fazer alterações na sua conta.

Body:

{

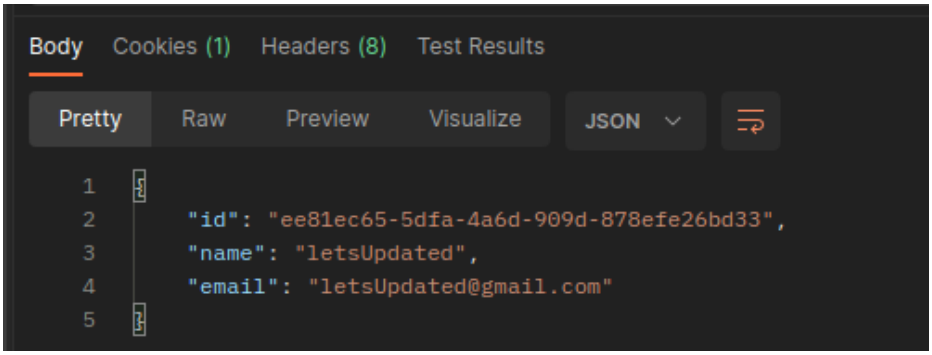
"name":"letsUpdated",

"email":"letsUpdated@gmail.com",

"password":"111"

}

Response:

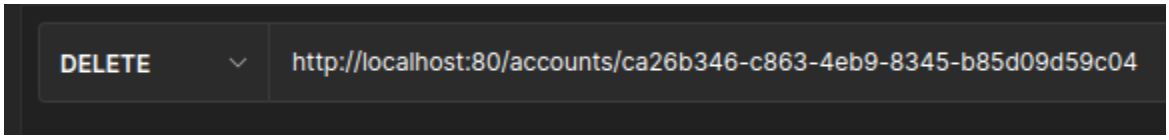


Acessando a rota GET mais uma vez, pode-se notar que o usuário foi atualizado com sucesso:

Caso 5: DELETE (http://localhost:80/accounts/:id) – Deletar a conta de um usuário baseado no ID.

Optou-se por fazer o delete do usuário que foi criado, cujo id é:

ca26b346-c863-4eb9-8345-b85d09d59c04



Response:

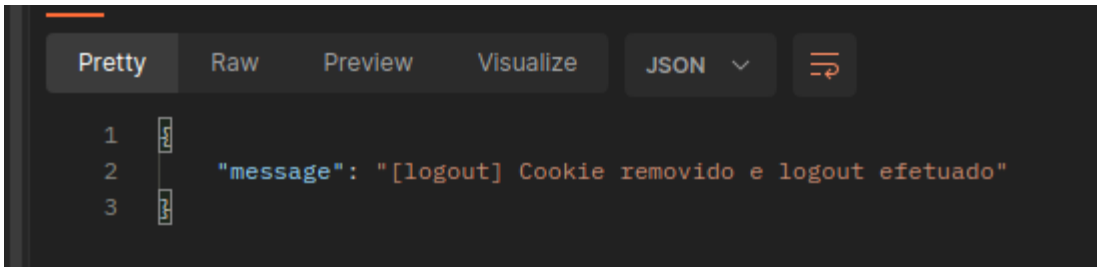
retorna os dados do usuário que foi deletado

Usando a rota GET nota-se que o usuário não está presente mais.

Por fim, utiliza-se a rota de logout para fazer o logout do usuário

POST (http://localhost:80/accounts/logout)

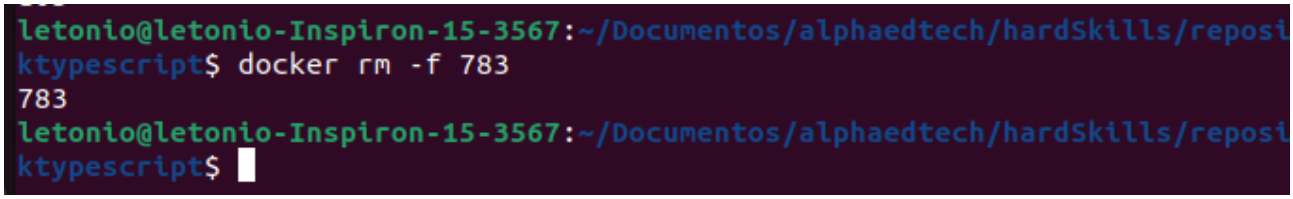
Response:



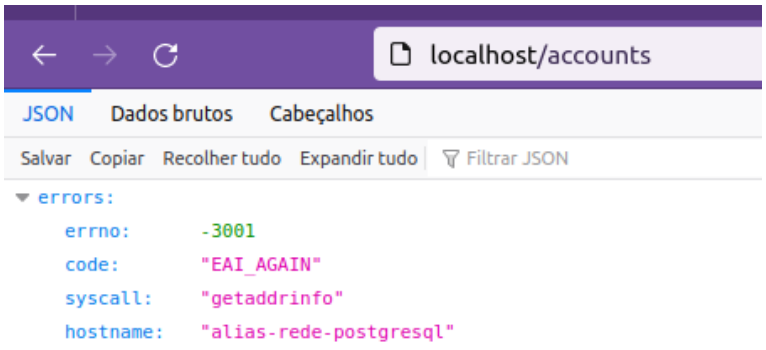
Vamos verificar a persistência dos dados.

Removendo o container do postgres, o comando abaixo força o container a parar e o exclui.

docker rm -f 783



Recarregando o navegador, nota-se que não tem mais dados, apenas uma mensagem de erro:

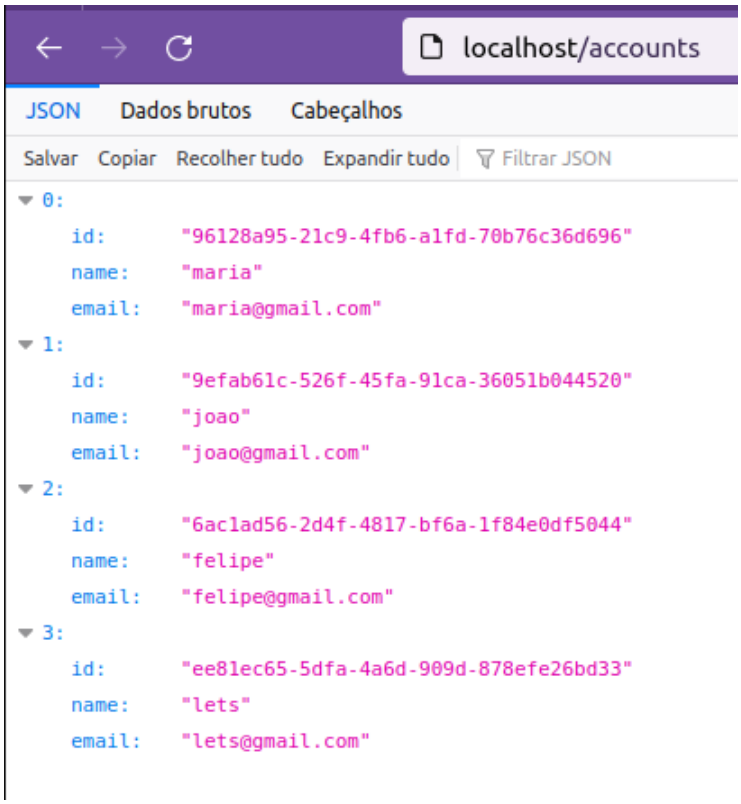


Vamos criar um novo container, passando as mesmas informações, sendo a mais relevante o volume, pois foi nele que as informações foram armazenadas da primeira vez.

docker run -d --name db-aula05 --network net-aula05 --network-alias alias-rede-postgresql -v vol-aula05:/var/lib/postgresql/data -e POSTGRES_USER=letsadmin -e POSTGRES_PASSWORD=1234 -e POSTGRES_DB=db-aula05 postgres

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05_ba
ktypescript$ docker run -d --name db-aula05 --network net-aula05 --network-alias alias-rede-postgresql -v vol-aula05:/var/lib/postgresql/data
-e POSTGRES_USER=letsadmin -e POSTGRES_PASSWORD=1234 -e POSTGRES_DB=db-aula05 postgres
37c2f53d6ea208c51361ee16772c6196a508687eb3cc5fb882fb9be5a9daf85e
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/Alpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05_ba
ktypescript$
```

Após executar o comando acima, podemos recarregar a página e novamente vamos visualizar as informações que tinham sido registradas:



Opcionalmente, a partir do terminal, podemos executar esse comando:

docker exec -it 37c2 psql -U letsadmin -d db-aula05

Com o cliente psql aberto, podemos usar \dt para ver as tabelas (nesse caso temos apenas uma accounts):

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/
ktypescript$ docker exec -it 37c2 psql -U letsadmin -d db-aula05
psql (15.2 (Debian 15.2-1.pgdg110+1))
Type "help" for help.

db-aula05=# \dt
          List of relations
 Schema |   Name   | Type  |   Owner
--------+----------+-------+-----------
 public | accounts | table | letsadmin
(1 row)

db-aula05=#
```

Aplicando a query “SELECT * FROM accounts;”, obtém-se o resultado a seguir:

```
db-aula05=# SELECT * FROM accounts;
                  id                  |  name  |      email       | password |         created_at         |         updated_at         |
ed_at | deleted
--------------------------------------+--------+------------------+----------+----------------------------+----------------------------+
------+---------
 96128a95-21c9-4fb6-a1fd-70b76c36d696 | maria  | maria@gmail.com  | 222      | 2023-05-21 16:16:27.299123 |                            |
      | f
 9efab61c-526f-45fa-91ca-36051b044520 | joao   | joao@gmail.com   | 333      | 2023-05-21 16:16:27.299123 |                            |
      | f
 6ac1ad56-2d4f-4817-bf6a-1f84e0df5044 | felipe | felipe@gmail.com | 444      | 2023-05-21 16:16:27.299123 |                            |
      | f
 ee81ec65-5dfa-4a6d-909d-878efe26bd33 | lets   | lets@gmail.com   | 123      | 2023-05-21 16:16:27.299123 | 2023-05-21 22:01:29.677462 |
      | f
(4 rows)

db-aula05=#
```

Nota-se que são as mesmas informações da última imagem do navegador. Além disso, percebe-se que o campo updated_at de lets não é null, pois ele foi atualizado para letsUpdated e, em seguida, modificado para lets novamente.

Usa-se \q para sair o cliente psql.

```
db-aula05=# \q
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/reposi
ktypescript$
```

Para parar os container pode-se utilizar “docker stop container-id”.

docker inspect db-aula05 | grep IPAddress permite encontrar facilmente o IP do container.

```
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/A
lpha-edtech-cycle01/Module15-Docker/Aula05/Q3_aula05_backtypescript$ docker inspect db
-aula05 | grep IPAddress
            "SecondaryIPAddresses": null,
            "IPAddress": "",
                    "IPAddress": "172.25.0.2",
letonio@letonio-Inspiron-15-3567:~/Documentos/alphaedtech/hardSkills/repositorioHard/A
```