

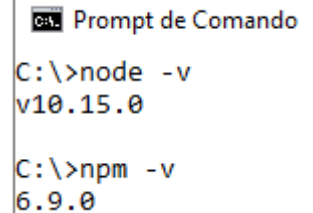
Objetivo: Fazer um servidor Node Express para persistir os dados numa tabela do SGBD PostgreSQL.

Utilize os arquivos em anexo e os passos a seguir para fazer a atividade.

Observações: antes de começar a atividade você precisa ter instalado o Node.js. Para checar isso, acesse o CMD do Windows e digite os comandos ao lado para checar as versões – as versões podem ser diferentes.

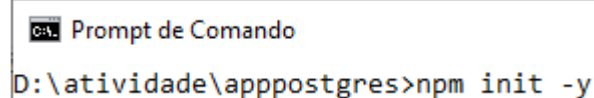
Também é necessário ter o SGBD PostgreSQL. Nele você precisa criar um BD de nome **bdaula** e usar as cláusulas a seguir para criar a tabela **tbaluno**:

```
drop table if exists tbaluno;  
create table if not exists tbaluno(  
    idaluno serial not null primary key,  
    nome varchar(50) not null,  
    idade integer not null  
);
```



```
C:\> node -v  
v10.15.0  
  
C:\> npm -v  
6.9.0
```

1 – Criar um projeto: crie uma pasta de nome **apppostgres** no seu computador. O nome da pasta não é importante, porém siga esse nome para manter a compatibilidade com a explicação. Todo o código da sua aplicação ficará na pasta **apppostgres**. Use o comando **npm** (node package manager) **init** para criar o projeto Node na pasta **apppostgres**:



```
C:\> Prompt de Comando  
D:\atividade\apppostgres> npm init -y
```

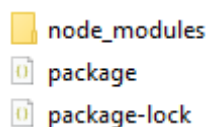
O comando irá criar o arquivo **package.json** na pasta **apppostgres**. Esse arquivo possui as configurações de bibliotecas do nosso projeto.

2 – Instalar o pacote Express: o Express (<https://expressjs.com/pt-br/>) é um framework para ajudar no tratamento de **request** (requisições do cliente/navegador) e requisições HTTP.

Na pasta **apppostgres** digite o seguinte **npm** para instalar. A instalação do Express pode ser **npm install** ou **npm i** ou **npm add**:

```
D:\atividade\apppostgres> npm install express
```

Após instalar o Express, a pasta **apppostgres** terá o seguinte conteúdo:



```
node_modules  
package  
package-lock
```

3 – Instalar o body-parser: o objetivo desse projeto é enviar parâmetros do cliente/navegador para o servidor Node. O pacote **body-parser** nos ajuda a receber estes parâmetros no servidor Node, e também é capaz de converter o corpo da requisição para vários formatos, incluindo JSON. O body-parser nos ajuda principalmente quando faremos solicitações usando os métodos POST (insert) e PUT (update).

Na pasta **apppostgres** digite o comando npm para instalar:

```
D:\atividade\apppostgres> npm install body-parser
```

4 – CORS (Cross-Origin Resource Sharing): o servidor web tipicamente não aceita requisições oriundas de outros domínios. Para permitir CORS no Node Express precisaremos instalar o pacote para CORS. Na pasta `apppostgres` digite o comando npm para instalar:

```
D:\atividade\apppostgres>npm install cors
```

5 – SGBD PostgreSQL: para acessar o SGBD PostgreSQL precisaremos instalar o pacote `node-postgres` (<https://node-postgres.com/>). Essa biblioteca possui o código para fazermos a conexão com o banco de dados.

Na pasta `apppostgres` digite o comando npm para instalar:

```
D:\atividade\apppostgres>npm install pg
```

6 – Pasta para colocar os arquivos do projeto: crie uma pasta de nome `src` dentro da pasta `apppostgres`. O nome da pasta não é importante, porém siga esse nome para manter a compatibilidade com a explicação. Dentro da pasta `src` colocaremos os arquivos do projeto/aplicação.

Copie os arquivos da pasta `src-postgres` em anexo, para a pasta `src` que você acabou de criar.

7 – Arquivo `index.js` de configuração do servidor: nesse arquivo colocamos o código para inicializar o nosso servidor e as definições de rotas. As instruções a seguir são usadas para importar as bibliotecas e fazer o servidor escutar a porta `3100`:

```
// importa o módulo express e coloca na variável express
const express = require("express");
// importa o módulo body-parser
const bodyParser = require('body-parser');
// importa o módulo CORS
var cors = require('cors');
// cria a aplicação chamando a função express()
const app = express();
// indica que usaremos o body-parser para ele entender quando enviarmos parâmetros no formato JSON
app.use(bodyParser.json());
// para ele decodificar parâmetros enviados via URL, no formato key=value e separados por &
app.use(bodyParser.urlencoded({ extended: true }));
// para aceitar CORS
app.use(cors());
// para deixar o seu servidor rodando na porta 3100 http://localhost:3100/
app.listen(3100, () => {
  console.log("Servidor rodando na porta 3100...");
});
```

Lembre-se que para testar o código você precisa subir o servidor, então, no CMD, acesse a pasta que possui o arquivo `index.js` e digite o comando

`node index.js` ou `nodemon index.js`

Este último só é possível se você tiver instalado o nodemon: `npm install nodemon`

8 – CRUD (Create, Read, Update, Delete): codificar o CRUD na tabela `tbaluno` implica em ter funções para fazer as 4 operações básicas na tabela (`insert`, `select`, `update` e `delete`). Para manter o código mais organizado colocamos esse código no arquivo `clausulas.js` (na pasta `src`) para não sobrecarregar o arquivo `index.js`.

A seguir tem-se o código para importar a classe `Pool` do módulo `pg` (da biblioteca `node-postgres`):

```
const Pool = require('pg').Pool
// cria um pool de conexão, isso evita ter de abrir um cliente e fechar a cada consulta
const pool = new Pool({
  host: 'localhost',
  database: 'bdaula',
  user: 'postgres',
  password: '123',
  port: 5432,
});
```

Como exemplo, a seguir tem-se o código da função que retorna todos os registros da `tbaluno`. A função `getAlunos` recebe como parâmetro um objeto com os dados da requisição do cliente/navegador - tal como, parâmetros - e um objeto com os dados a serem devolvidos (`response`) para o cliente/navegador:

```
const getAlunos = (request, response) => {
  pool.query('select * from tbaluno order by nome', (error, results) => {
    if (error) {
      throw error;
    }
    response.status(200).json(results.rows);
  });
};
```

Para que as funções do módulo `clausulas.js` possam ser importadas no módulo `index.js`, elas precisam ser exportadas:

```
module.exports = {
  getAlunos,
  getAlunosById,
  createAluno,
  deleteAluno,
  updateAluno
};
```

No arquivo `index.js` precisaremos importar o módulo `clausulas` fazendo:

```
const bd = require('./clausulas');
```

Precisaremos também incluir a rota para cada função exportada:

```
// a função getAlunos foi mapeada para a URL http://localhost:3100/alunos
app.get("/alunos", bd.getAlunos);
// o parâmetro id será passado na URL da seguinte forma http://localhost:3100/alunos/2
app.get('/alunos/:id', bd.getAlunosById);
// a diferença entre a chamada do select e insert é o método POST
app.post('/alunos', bd.createAluno);
// para fazer um delete precisamos usar o método DELETE
```

```
app.delete('/alunos/:id', bd.deleteAluno);  
// para fazer um update precisamos usar o método PUT  
app.put('/alunos/:id', bd.updateAluno);
```

As requisições do cliente para o servidor podem ser do tipo GET, POST, PUT e DELETE. O método implica no tipo de operação que será executada na tabela do SGBD:

- GET será para fazer um **select** na tbaluno;
- POST será para fazer um **insert** na tbaluno;
- PUT será para fazer um **update** na tbaluno;
- DELETE será para fazer um **delete** na tbaluno;

Para testar as operações você precisa estar com o servidor rodando e no CMD digitar os seguintes comandos. O cURL (Cliente URL) é uma ferramenta de linha de comando para transferir dados usando vários protocolos, aqui será usado o HTTP.

- Para testar o select para todos os registros: `curl http://localhost:3100/alunos`
- Para fazer o select na tabela e retornar o registro que possui idaluno igual a 1:
`curl http://localhost:3100/alunos/2`
- Para fazer o insert: `curl -d "nome=Lucas Dias&idade=20" http://localhost:3100/alunos`
- Para fazer o delete no registro que possui idaluno igual a 1: `curl -X DELETE http://localhost:3100/alunos/1`
- Para fazer o update no registro que possui idaluno igual a 3: `curl -X PUT -d "nome=Ana Maria&idade=22" http://localhost:3100/alunos/3`