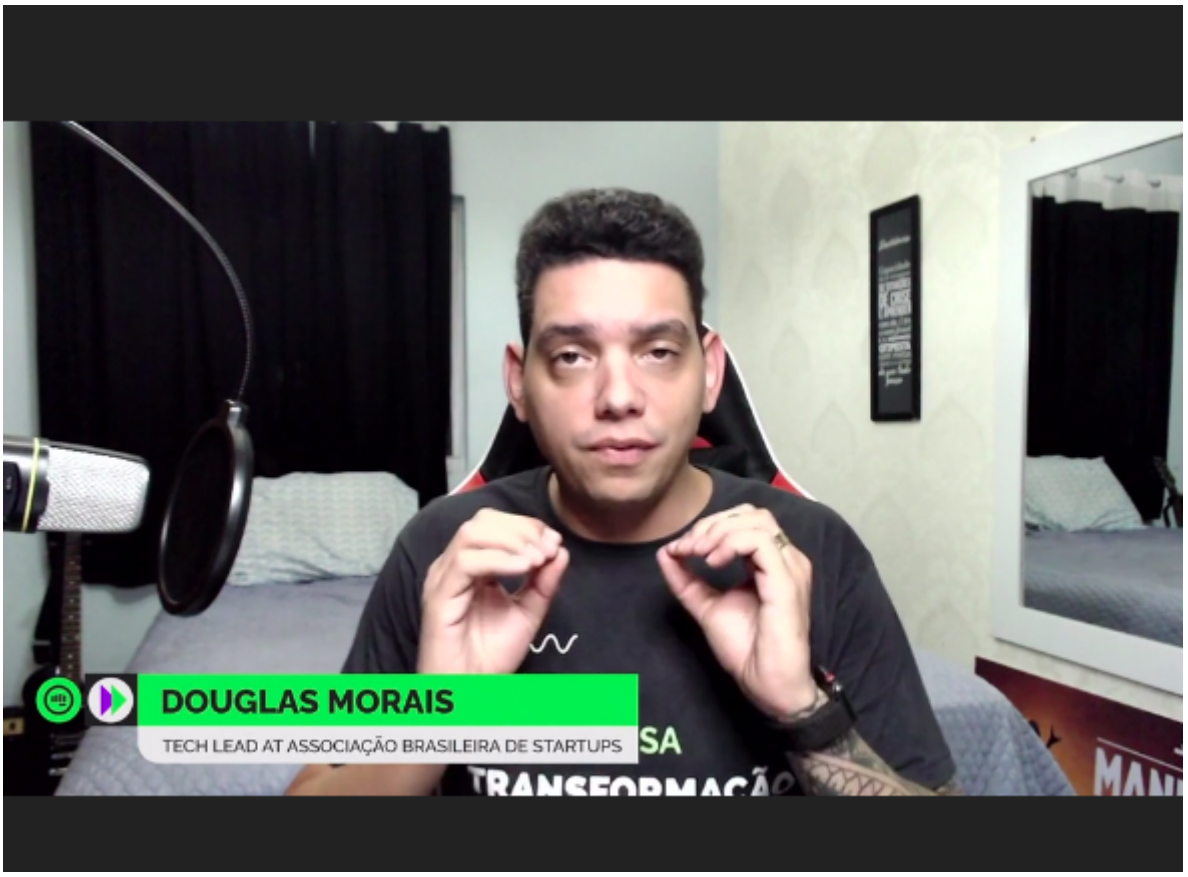
NODEJS - HTTP GET e HTTP POST



Pessoal, a ideia deste repositório é repetir o processo de criação do aplicativo de agendamento criado na aula do professor Douglas Morais para a semana 5 do Hiring Coders #3 2022.

Este repositório foi criado ao mesmo tempo que todas as ações foram sendo registradas na lista abaixo. Cada fase possui uma branch com seu nome o que esperamos que facilite a execução e eventuais reversões em função de problemas, se contudo perder todo o trabalho feito até a faze anterior.

Pedimos que compartilhem caso inconsistências ou oportunidades de melhoria sejam identificadas.
Até o momento foram criadas as branches abaixo. A master possui o código completo.

- master
- phase02-addServerHttpUsingCommonJS
- phase03-addNodemonAndExpress
- phase04-addDatabase
- phase05-addCreateUserFeature
- * phase06-addUserController

- Repositório com o projeto do professor:
  https://github.com/mrdouglasmorais/sistemadeagendamento

  Ao longo deste tutorial **usaremos a palavra comissionar com o sentido de** executar o comando **git commit**.
  A palavra comissionar tem como um dos seus possíveis significados o de entregar, usada de forma consagrada na indústria naval na fase de entrega de grandes projetos de alta complexidade como navios, plataformas de exploração de petróleo etc.
  O comissionamento também ocorre em fases intermediárias de um projeto naval, validando as entregas intermediárias até que ocorra o comissionamento final.
  Por esse motivo estamos propondo a adoção desse termo para a tradução de commit.

  Descritivo de desenvolvimento:
- Criar a estrutura inicial do projeto:
  - mkdir bookingSystemNodeJSHttp
    Entrar na pasta
  - cd hiringCoders2022-nodeHttp
  - git init
  - touch README.md
  - touch .gitignore
  - mkdir src
    Entrar na pasta
  - cd src
  - touch app.js
  - touch routes.js
  - touch server.js
    Sair da pasta
  - cd ..

- Inicializar o módulo NodeJS. Ao usarmos a flag '-y' as perguntas foram respondidas automaticamente e o arquivo package.json será criado.
  - npm init -y

- Comissionar fase 1: Estrutura de arquivos inicial criada.
    - git add .
    - git commit -m "build: add a basic file structure of a node js project"

- Criar uma nova branch:
    - git checkout -b phase02-addServerHttpUsingCommonJS

- Instalar o express. Este será o framework usado para desenvolver o servidor.
    - npm i express
    - Resultado esperado: criação da pasta node_modules e do arquivo package-lock.json.

- Com base nos arquivos disponibilizados neste repositório:
    - Desenvolver os arquivos:
        - touch app.js
        - touch routes.js
        - touch server.js

- Testar a fase 2: Foi criado um servidor http usando NodeJS e express. A aplicação é capaz de realizar as seguintes funções:
    - Ser inicializada:
        - node src/server.js
    - Ser acessada pela URL a seguir:
        - http://localhost:3333/api-test
    - Encerrar o servidor: digitar CONTROL + C

- Comissionar fase 2:
    - git add .
    - git commit -m "feat: add the basic structure of a node js server"
    - git checkout master
    - git merge phase02-addServerHttpUsingCommonJS

- Criar uma nova branch para a próxima fase:
    - git checkout -b phase03-addNodemonAndExpress

- Instalar o nodemon e o sucrase para aumentar a produtividade.
  O nodemon : automaticamente detecta os arquivos alterados e reinicia a aplicação.
  O sucrase permite desenvolvermos a aplicação node em ES6 (ECMAScript6)
    - npm install sucrase -D
    - npm install nodemon -D
    - Criar o arquivo nodemon.json:
        - { "execMap": {"js": "node -r sucrase/register" }}

    - Colocar no script do package.json:

        - "dev": "nodemon src/server",
- Testar:
    - Inicializar o servidor:
        - npm run dev
    - Acessar a URL a seguir:
        - http://localhost:3333/api-test
    - Encerrar o servidor: digitar CONTROL + C

- Comissionar adição do sucrase e do nodemon:
    - git add .
    - git commit -m "build: add sucrase and nodemon"

- Refatorar o código de CommonJS para ECMAScript6:
    - Inicializar o servidor. Quando a refatoração estiver completa o nodemon conseguirá inicializar o servidor:
        - npm run dev
    - Alterar as instruções de importação e exportação de CommonJS para ECMAScript 6 nos arquivos:
        - app.js
        - server.js
        - routes.js

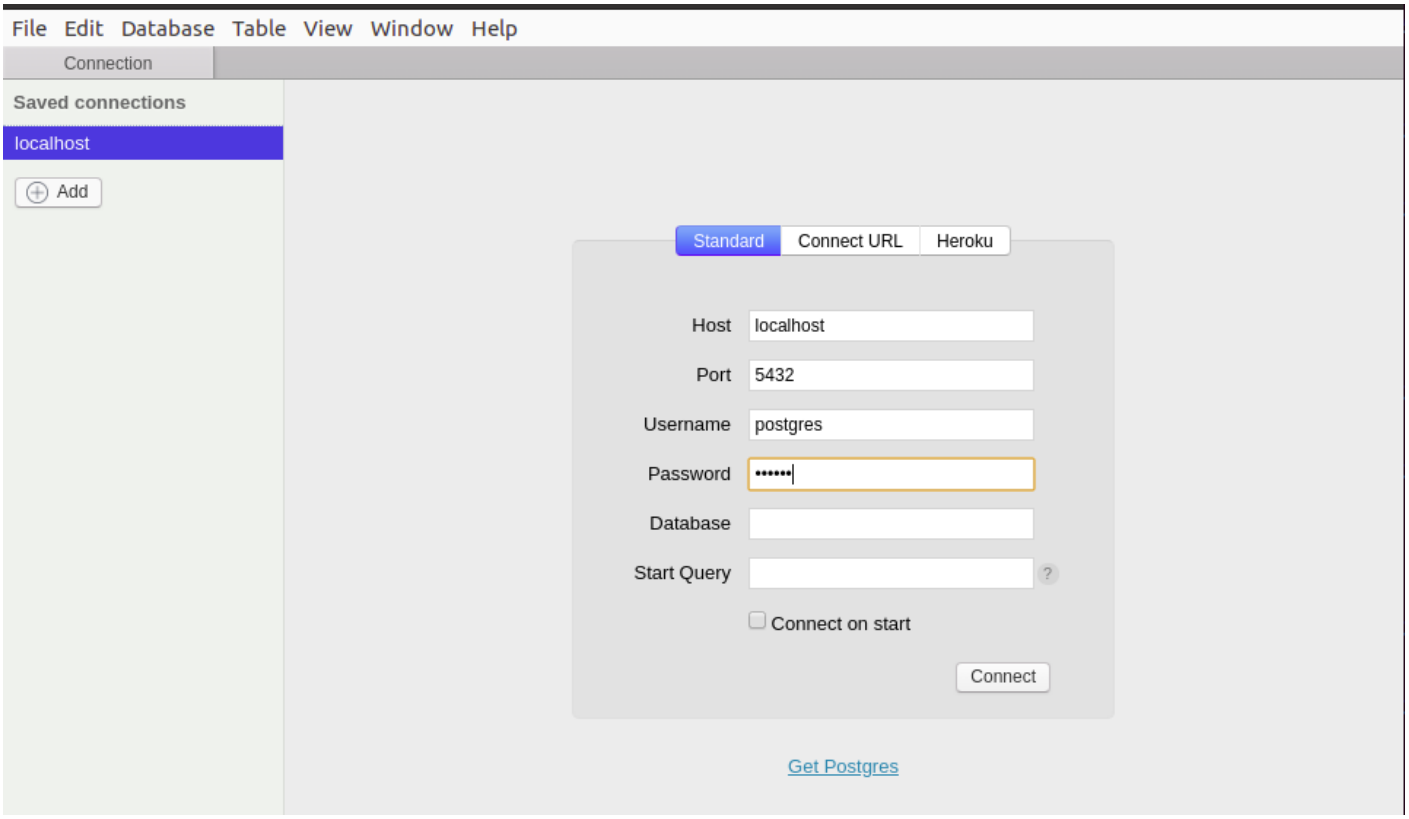
- Testar todas as funcionalidades antes de iniciar o desenvolvimento da próxima fase.

- Comissionar os arquivos alterados ainda com as linhas CommonJS comentadas para evidenciar as alterações:
    - git add .
    - it commit -m "refactor: change imports and exports from CommonJS to ECMAScript 6"
    - git checkout master
    - git merge phase03-addNodemonAndExpress

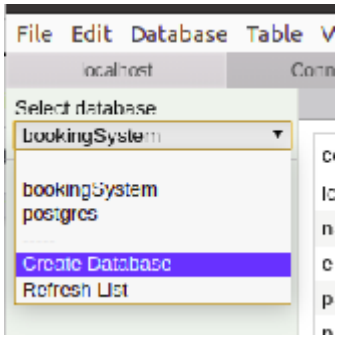
- Criar uma nova branch para a próxima fase:
    - git checkout -b phase04-addDatabase

- Instalar o Docker e o banco de dados Postgres:
    - Instalar Docker engine
    - Instalar imagem do Postgres no Docker:
        - sudo docker run --name database -e POSTGRES_PASSWORD=docker -p 5432:5432 -d postgres
    - Listar containers para visualizar a imagem do Postgres com nome ‘database’:
        - sudo docker ps -a
    - Dar start no database, caso não esteja rodando:

        - sudo docker start database
    - Parar o database:
        - sudo docker stop database
    - Remover a imagem do database:
        - sudo docker rm database
- Instalar o Postbird:
    - sudo snap install postbird
- Conectar o Postbird à imagem do Postgres instalada no Docker:
    - Password: docker



- Criar um banco de dados chamado ‘bookingSystem’ usando o Postbird.



- Instalar o ORM:
  - npm i sequelize
  - npm i --save-dev sequelize-cli
- Instalar o driver do Postgres - lista de drivers:
  - npm i pg pg-hstore
- Criar o arquivo ‘.sequelizerc’ na pasta raíz do projeto. Ele permite mudar a localização padrão das pastas, inclusive do arquivo de configuração da conexão com o banco de dados que por padrão fica em ‘config/config.json’:
  - touch .sequelizerc
- Preencher o arquivo .sequelizerc.
- Criar o arquivo ‘src/config/database.js’:
  - mkdir src/config
  - touch src/config/database.js
- Testar todas as funcionalidades antes de iniciar o desenvolvimento da próxima fase.

- Comissionar fase 04:
  - git add .
  - git commit -m "build: add orm sequelize and configuration file .sequelizerc"
  - git checkout master
  - git merge phase04-addDatabase

- Criar uma nova branch para a próxima fase:
  - git checkout -b phase05-addCreateUserFeature
- Testar todas as funcionalidades antes de iniciar o desenvolvimento da próxima fase.

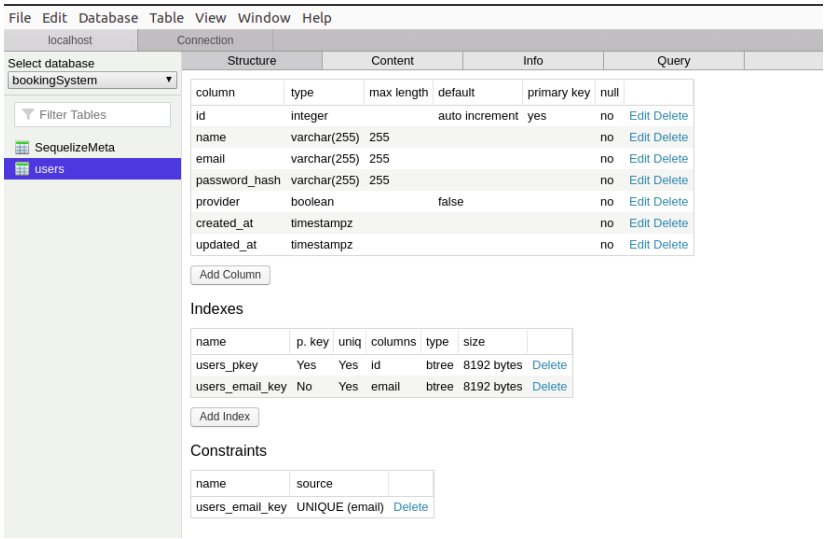
- Criar o arquivo ‘src/app/models/index.js’:
  - mkdir src/app/
  - mkdir src/app/models/
  - touch src/app/models/index.js
- Configurar ‘src/app/models/index.js’.

- Criar o arquivo src/app/models/User.js
    - touch src/app/models/User.js
- Configurar o arquivo User.js

- Criar o arquivo ‘src/database/index.js’:
    - touch src/database/index.js
- Configurar ‘src/database/index.js’.

- Criar a migration ‘create-users’.
    - npx sequelize migration:create --name=create-users
    - Abaixo um exemplo do nome do arquivo criado:
        - 20220531041044-create-users.js
- Configurar a migration para criar a tabela ‘users’.
- Executar as migrations para criar as tabelas no banco de dados:
    - npx sequelize db:migrate
- Desconsiderar se tudo ocorrer como esperado.
    - Desfazer:
        - npx sequelize db:migrate:undo
        - npx sequelize db:migrate:undoAll

- Verificar se a tabela users foi criada no banco de dados usando Postbird:



- Testar a criação de novos registros usando o Postbird:
    - Excluir todos os registros criados anteriormente, se houver:
        - DELETE FROM users;

    - Inserir um novo registro:
        - INSERT INTO users (id,name,email,password_hash,created_at,updated_at) VALUES (DEFAULT, 'newUser', 'userEmail', 'userPassword', now(), now());
- Configurar uma nova rota em routes.js:

- import UserController from './app/controllers/UserController';
- import Database from './database/index';
- Users cria registro dentro da função call back de routes:

```
routes.post('/users', async(req, res) => {
        const user = await User.create({
           name: "user-name-1",
           email: "user1@server.com",
           password_hash: "123456"
        });
        return res.json(user);
});
```

- Testar todas as funcionalidades antes de iniciar o desenvolvimento da próxima fase.

- Comissionar fase 05:
  - git add .
  - git commit -m "feat: add create user feature using mock user from routes.js"
  - git checkout master
  - git merge phase05-addCreateUserFeature

- Testar todas as funcionalidades antes de iniciar o desenvolvimento da próxima fase.

- Criar uma nova branch para a próxima fase:
  - git checkout -b phase06-addUserController