

The Proffy logo is displayed in a large, white, sans-serif font against a solid purple background. The background is decorated with various green geometric shapes, including circles, squares, and crosses, some of which are semi-transparent.

Deploy NLW

Nesse documento, nós vamos ver como fazer o deploy da aplicação Proffy, desenvolvida durante a Next Level Week da [Rocketseat](#), de maneira gratuita, utilizando o Heroku e o Netlify.

Deploy do servidor da aplicação

Primeiro passo - Configurando o banco de dados

Preparando a aplicação

Configurando o PostgreSQL

Preparando nossa aplicação

Testando nossa aplicação localmente

Criando nossas tabelas

Configurações avançadas

Segundo passo - Enviando nossa aplicação ao GitHub

Terceiro passo - Enviando a aplicação ao Heroku

Preparando o deploy

Quarto passo - Preparando o banco de dados

Criando a instância do banco de dados

Configurando credenciais do banco de dados

Finalizando o deploy do servidor

Executando as migrations

Deploy do Front-end Web da aplicação

Primeiro passo - Conectando a API

Segundo passo - Enviando nossa aplicação ao GitHub

Terceiro passo - Enviando a aplicação ao Netlify

Criando uma nova aplicação

Finalizando o Deploy

Deploy do servidor da aplicação

Nosso primeiro passo vai ser fazer o upload do nosso servidor, é ele que irá conectar com nosso banco de dados e salvar os dados que serão utilizados pelo nosso frontend.

Primeiro passo - Configurando o banco de dados

Se você acompanhou a Next Level Week, você com certeza sabe que nós utilizamos o SQLite, mas aqui não estaremos utilizando ele. Dessa vez, para levar nosso conhecimento um passo a frente, estaremos configurando o PostgreSQL na nossa aplicação.

Preparando a aplicação

Está pronto? Então vamos lá! Para começar, vamos abrir nosso projeto no VSCODE, navegando até a pasta `server` pelo seu terminal e executando o comando `code .`.

O nosso primeiro passo, será remover tudo que não iremos utilizar agora relacionados ao SQLite, para isso estaremos excluindo nosso arquivo `database.sqlite` e removendo a biblioteca `sqlite3` com o seguinte comando:

```
// Utilizando NPM npm uninstall sqlite3 // Utilizando Yarn yarn  
remove sqlite3
```

[Shell](#) ▾

Configurando o PostgreSQL

Agora que já estamos preparados para prosseguir configurando, adicionaremos a biblioteca do PostgreSQL, da seguinte forma:

```
npm install pg // Ou yarn add pg
```

[Shell](#) ▾

Feito isso, você já deve estar se perguntando "E agora, como irei me conectar com o banco de dados que eu possuía?". Pois então, nós não iremos! Como iremos utilizar um novo banco de dados, iremos começar configurando a conexão com ele, então iremos partir editando nosso arquivo `knexfile.ts` na raiz do nosso projeto.

Caso você tenha acompanhado a NLW, ele deverá estar da seguinte forma:

```
import path from 'path' module.exports = { client: 'sqlite3',  
connection: { filename: path.resolve(__dirname, 'src', 'database',  
'database.sqlite'), }, migrations: { directory:  
path.resolve(__dirname, 'src', 'database', 'migrations') },  
useNullAsDefault: true }
```

[TypeScript](#) ▾

Agora vamos alterar os dados que temos nesse arquivo para começar a parecer com o que teremos em produção. Primeiro vamos alterar a chave `client` para possuir o valor `pg`. Além disso removemos completamente a linha 6, e alteramos ela pelo seguinte:

```
host: 'localhost', user: 'postgres', password: 'docker', database: 'proffy',
```

JavaScript ▾

Mas CALMA! Não se preocupe, se você ainda não sabe de onde eu tirei esse host, user, password e database, vamos lendo até o final que você logo logo irá entender como irá funcionar!

O resultado final ficará dessa forma:

```
import path from 'path' import 'dotenv/config' module.exports = {
  client: 'pg', connection: { host: 'localhost', user: 'postgres',
  password: 'docker', database: 'proffy', }, migrations: { directory:
  path.resolve(__dirname, 'src', 'database', 'migrations') },
  useNullAsDefault: true }
```

TypeScript ▾

Bacana! Agora que configuramos nossa knexfile, vamos lembrar que temos um arquivo `connection.ts` dentro da nossa pasta `database` e vamos abrir ele também.

Aqui iremos fazer as mesmas alterações, alterando o client para `pg` e a connection para os dados que teremos no nosso banco de dados, ficando da seguinte forma:

```
import knex from 'knex' import 'dotenv/config' const db = knex({
  client: 'pg', connection: { host: 'localhost', user: 'postgres',
  password: 'docker', database: 'proffy', }, useNullAsDefault: true })
export default db
```

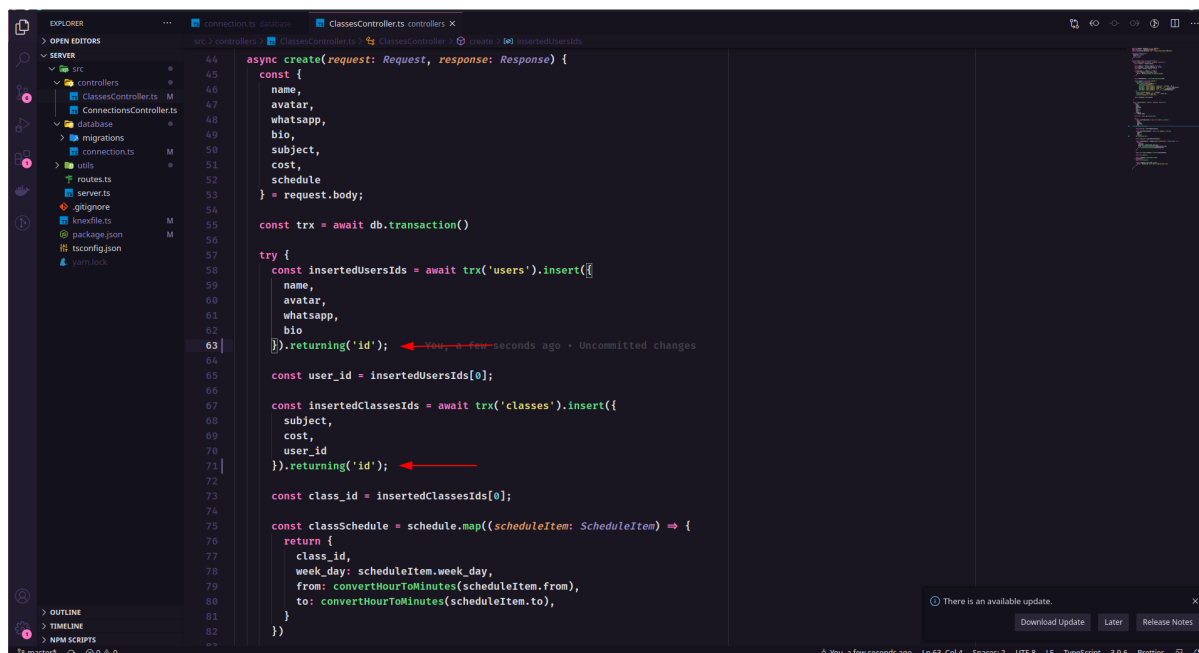
TypeScript ▾

Preparando nossa aplicação

Agora com essas mudanças, teremos algumas diferenças. Uma delas é que ao utilizar o PostgreSQL, no nosso insert de dados no banco de dados, ele não irá mais retornar os ids inseridos automaticamente, então teremos que fazer uma pequena alteração no nosso controller de classes, o `ClassesController`.

Dito isso, vamos já abrir esse arquivo e ir no nosso método create, e adicionar após o nosso insert, um trecho de código que irá indicar ao knex que ele deve retornar o id criado, que é: `.returning('id')`.

O resultado deve ficar da seguinte forma:



```

async create(request: Request, response: Response) {
  const {
    name,
    avatar,
    whatsapp,
    bio,
    subject,
    cost,
    schedule
  } = request.body;

  const trx = await db.transaction()

  try {
    const insertedUsersIds = await trx('users').insert({
      name,
      avatar,
      whatsapp,
      bio
    }).returning('id');
    const user_id = insertedUsersIds[0];

    const insertedClassesIds = await trx('classes').insert({
      subject,
      cost,
      user_id
    }).returning('id');
    const class_id = insertedClassesIds[0];

    const classSchedule = schedule.map((scheduleItem: ScheduleItem) => {
      return {
        class_id,
        week_day: scheduleItem.week_day,
        from: convertHourToMinutes(scheduleItem.from),
        to: convertHourToMinutes(scheduleItem.to),
      }
    })
  }
}

```

Caso você queira copiar, o código vai ficar assim:

```

const insertedUsersIds = await trx('users').insert({ name, avatar,
whatsapp, bio }).returning('id'); const user_id =
insertedUsersIds[0]; const insertedClassesIds = await
trx('classes').insert({ subject, cost, user_id }).returning('id');

```

TypeScript ▾

Agora faremos uma última alteração no nosso ClassesControllers, então abrimos esse arquivo, e na nossa query de busca de classes, nós vamos retirar TODAS as crases ``` dessa query, o resultado vai ficar assim:

```

ClassesController.ts controllers X
src > controllers > ClassesController.ts > ClassesController > index > classes > whereExists() callback
20     return response.status(400).json({
21       error: 'Missing filters to search classes'
22     })
23   }
24
25   const timeInMinutes = convertHourToMinutes(time)
26
27   const classes = await db('classes')
28     .whereExists(function() {
29     this.select('class_schedule.*')
30     .from('class_schedule')
31     .whereRaw('class_schedule.class_id = classes.id')
32     .whereRaw('class_schedule.week_day = ??', [Number(week_day)])
33     .whereRaw('class_schedule.from <= ??', [timeInMinutes])
34     .whereRaw('class_schedule.to > ??', [timeInMinutes])
35   })
36   .where('classes.subject', '=', subject)
37   .join('users', 'classes.user_id', '=', 'users.id')
38   .select(['classes.*', 'users.*'])
39
40   return response.json(classes)
41
42 }

```

Caso você queira copiar o código do select do resultado:

```

const classes = await db('classes') .whereExists(function() {
this.select('class_schedule.*') .from('class_schedule')
.whereRaw('class_schedule.class_id = classes.id')
.whereRaw('class_schedule.week_day = ??', [Number(week_day)])
.whereRaw('class_schedule.from <= ??', [timeInMinutes])
.whereRaw('class_schedule.to > ??', [timeInMinutes]) })
.where('classes.subject', '=', subject) .join('users',
'classes.user_id', '=', 'users.id') .select(['classes.*',
'users.*'])

```

TypeScript ▾

Prontinho, isso já deve ser suficiente para nossa aplicação funcionar com o PostgreSQL!

Testando nossa aplicação localmente

💡 Observação: Caso você não queira instalar o Docker para testar localmente, não há problema nenhum. Basta pular para a seção **Configurações avançadas**.

Eu sei, eu sei, você está doido pra ver o resultado disso né? Tá certo, então vou te ensinar uma coisinha. Mas para isso você vai precisar instalar o Docker Desktop, o Docker Desktop nada mais é que uma ferramenta que irá permitir que a gente rode um banco de dados postgres localmente na nossa máquina.

Caso você precise de ajuda, o Docker tem uma documentação completa ensinando a instalar ele nos principais sistemas operacionais:

<https://docs.docker.com/get-docker/>

Certo, conseguiu instalar e configurar tudo? Então vamos ao terminal, para isso nós rodaremos um comando que irá executar uma instância de um banco de dados na nossa máquina, e iremos tornar ela acessível a nossa aplicação na porta 5432, basta executar o seguinte código:

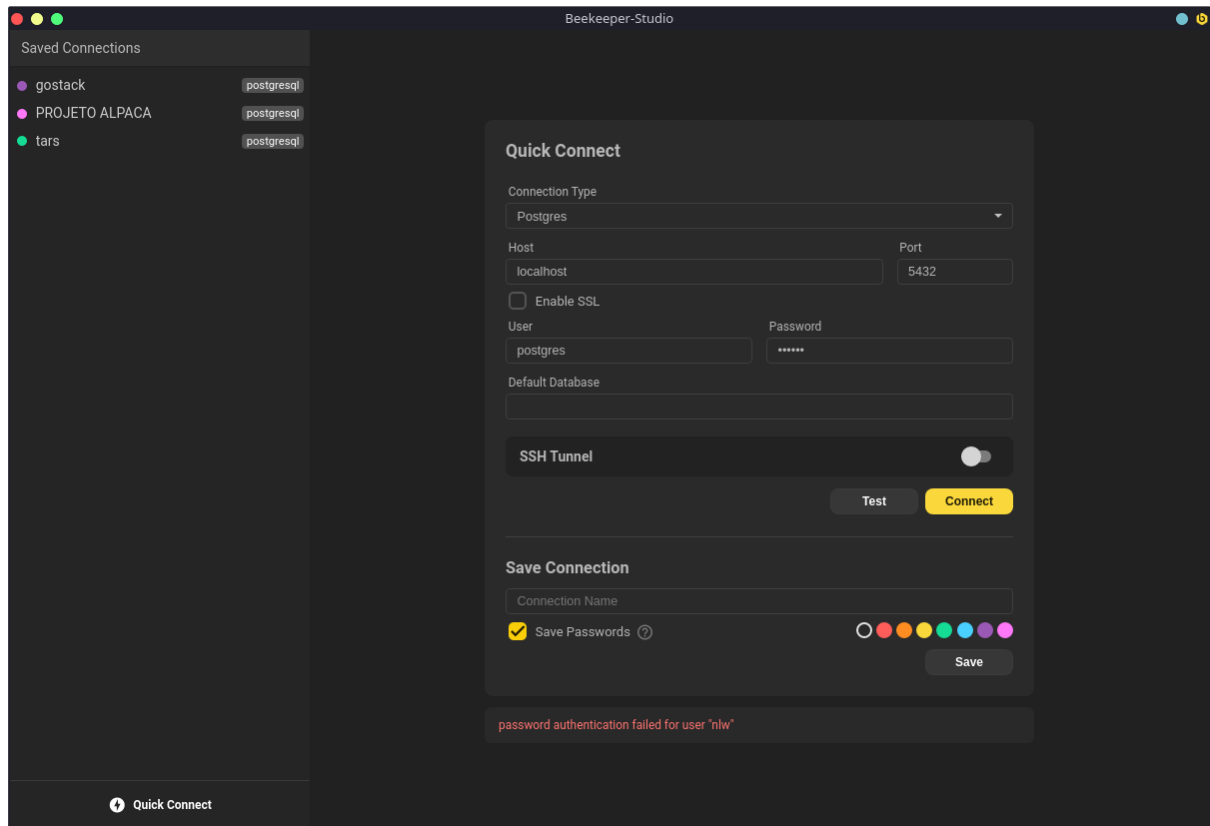
```
docker run --name nlw -e POSTGRES_PASSWORD=docker -p 5432:5432 -d postgres
```

Shell ▾

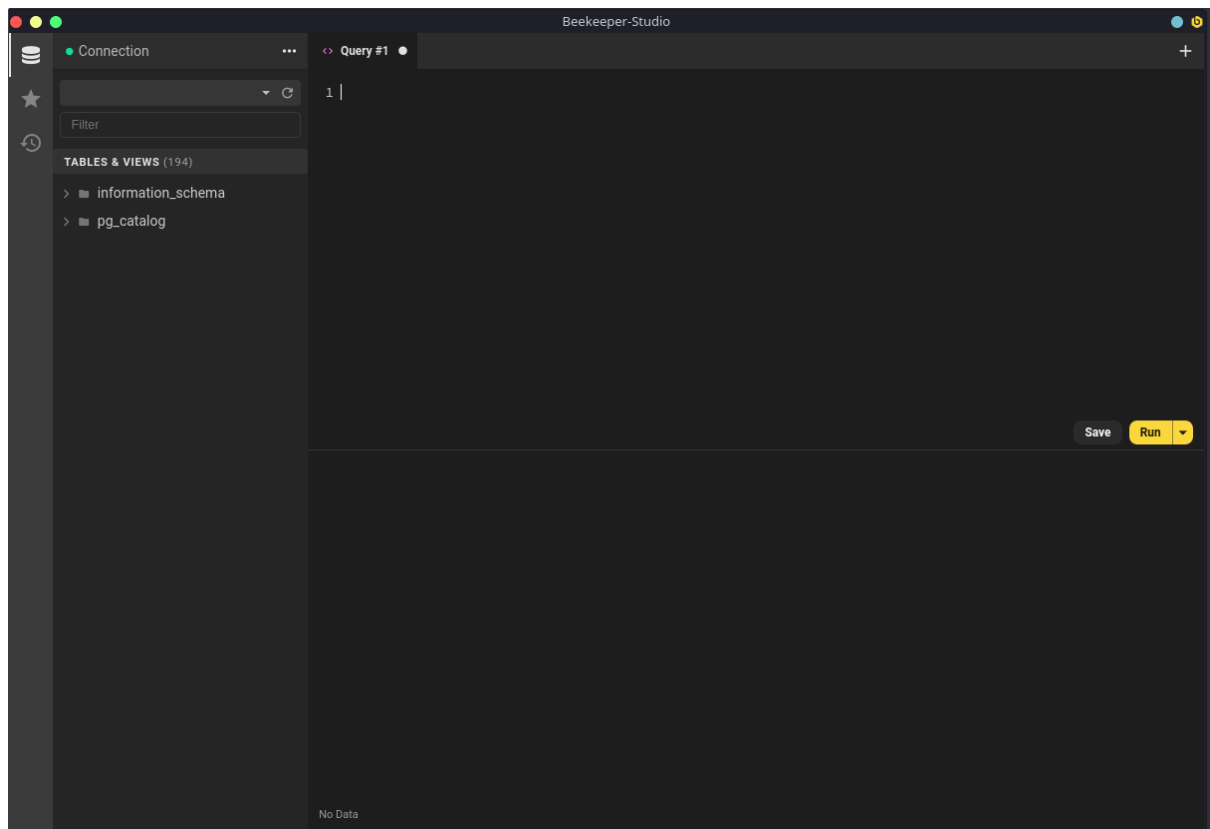
Isso irá criar um container na nossa máquina, configurado para que o postgres seja acessado pela porta 5432 e basta a gente ter um cliente de banco de dados compatível com o PostgreSQL que já conseguimos acessá-lo. No meu caso eu estou utilizando o **Beekeeper** no Ubuntu, mas você pode utilizar qualquer outra alternativa como Postbird ou DBeaver.

Lembra sobre quando eu disse que iríamos entender de onde viria os dados da nossa conexão? Pois executando aquele comando, nós poderemos acessar nosso banco com as seguintes informações:

- **Endereço:** localhost
- **Porta:** 5432
- **Usuário:** postgres (por padrão, o usuário será postgres)
- **Senha:** docker (Nós configuramos para a senha ser docker no comando executado)



Certo, agora basta clicar no botão de se conectar ao banco de dados no seu cliente favorito, e ele deve te jogar a uma página parecida com essa, onde você pode visualizar os bancos e as tabelas da sua instância do PostgreSQL.

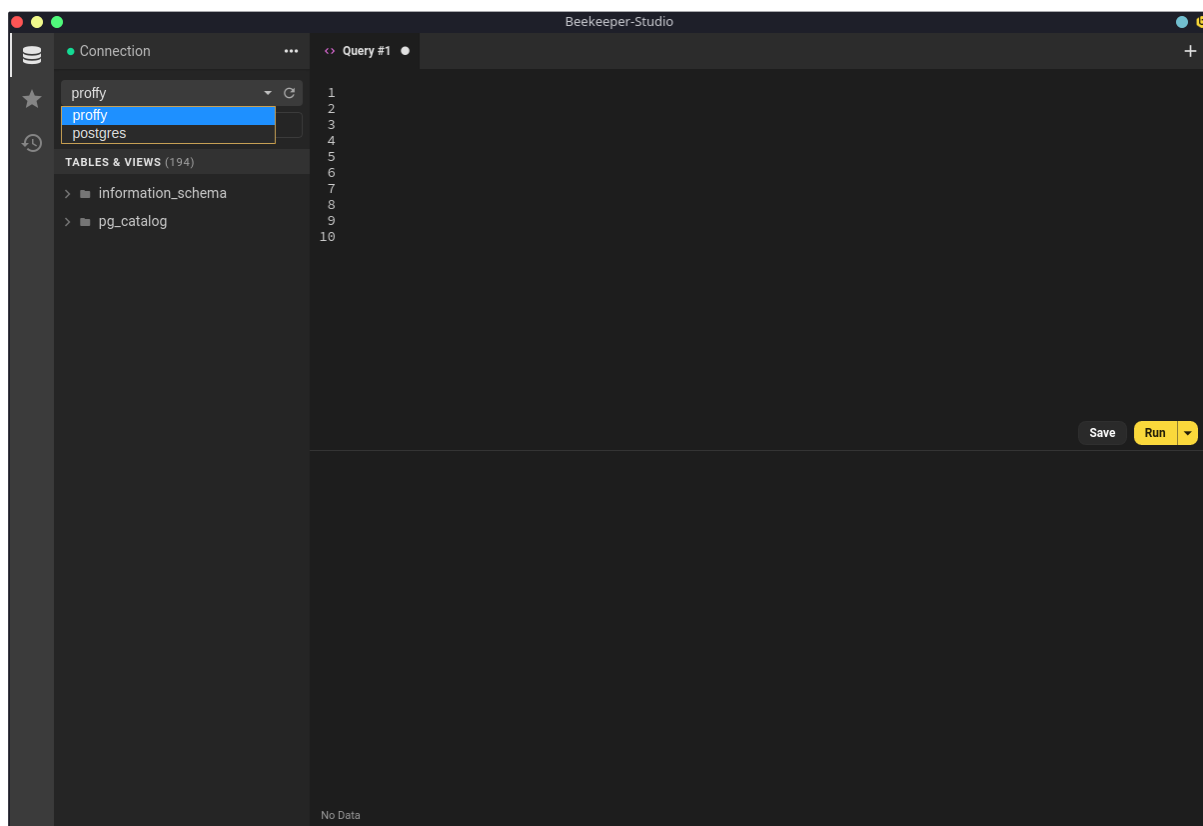


Por padrão, ele não cria nenhuma database pra gente, então vamos correr e criar a nossa. Eu vou dar o nome dela de proffy. Para isso, basta executar o seguinte comando:

```
CREATE DATABASE proffy;
```

SQL ▾

Agora basta selecionar essa database e pronto! Estamos prontos para continuar!



Criando nossas tabelas

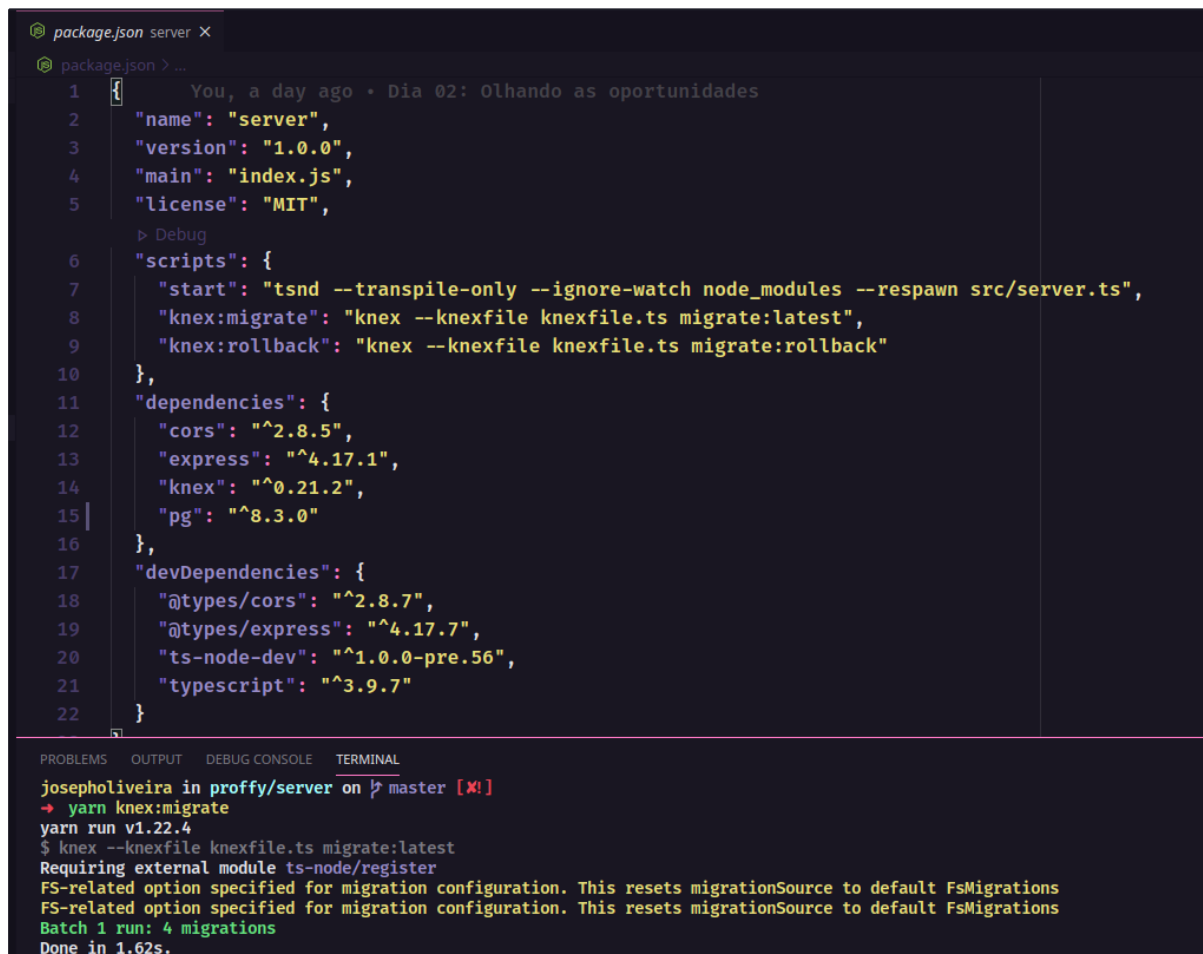
Claro, não podemos esquecer né? Agora que já configuramos nossa conexão com o banco de dados, iniciamos nossa instância do banco de dados e já preparamos nossa aplicação, nós precisamos fazer a parte mais importante, criar as tabelas!

Para isso, caso você tenha seguido 100% toda a criação das migrations e configuração do package.json e dos scripts de migrations, basta você executar no seu terminal o seguinte:

```
# No NPM npm run knex:migrate # No Yarn yarn knex:migrate
```

Shell ▾

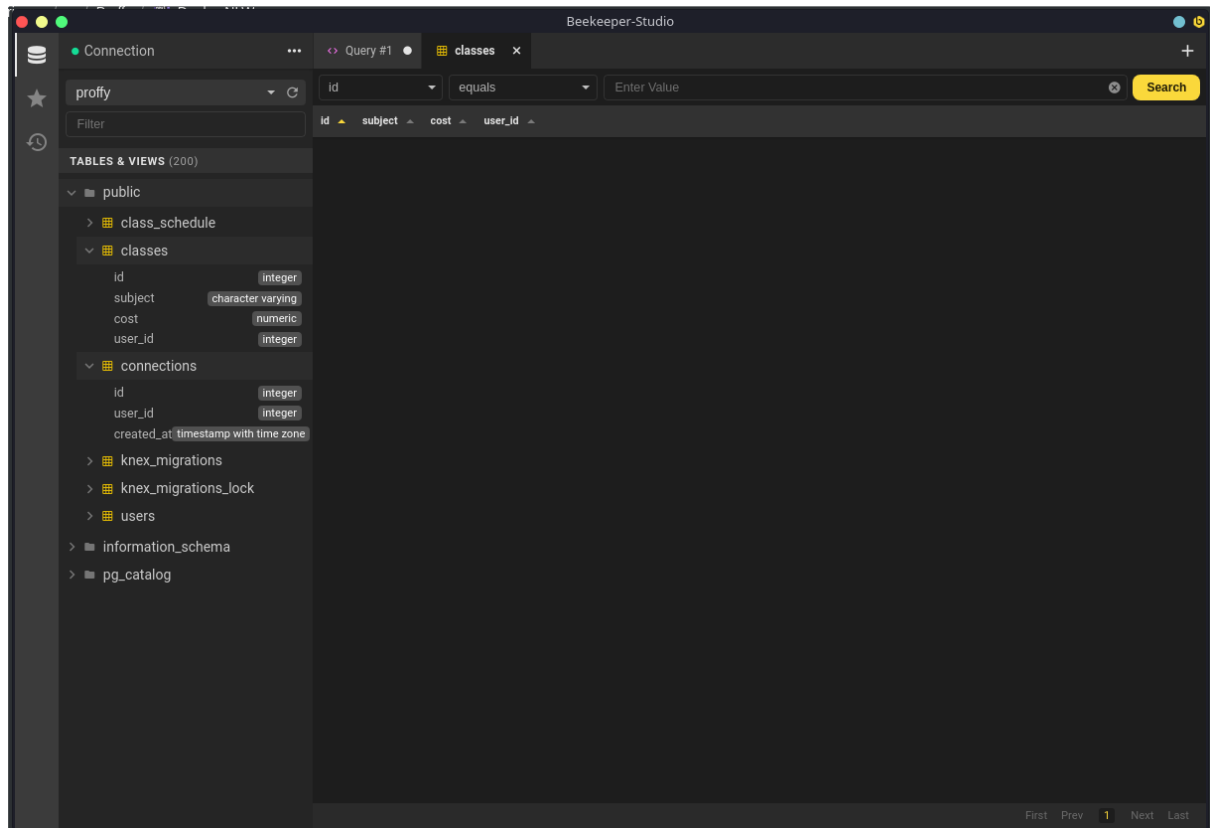
Você deve ter o seguinte resultado no seu terminal:



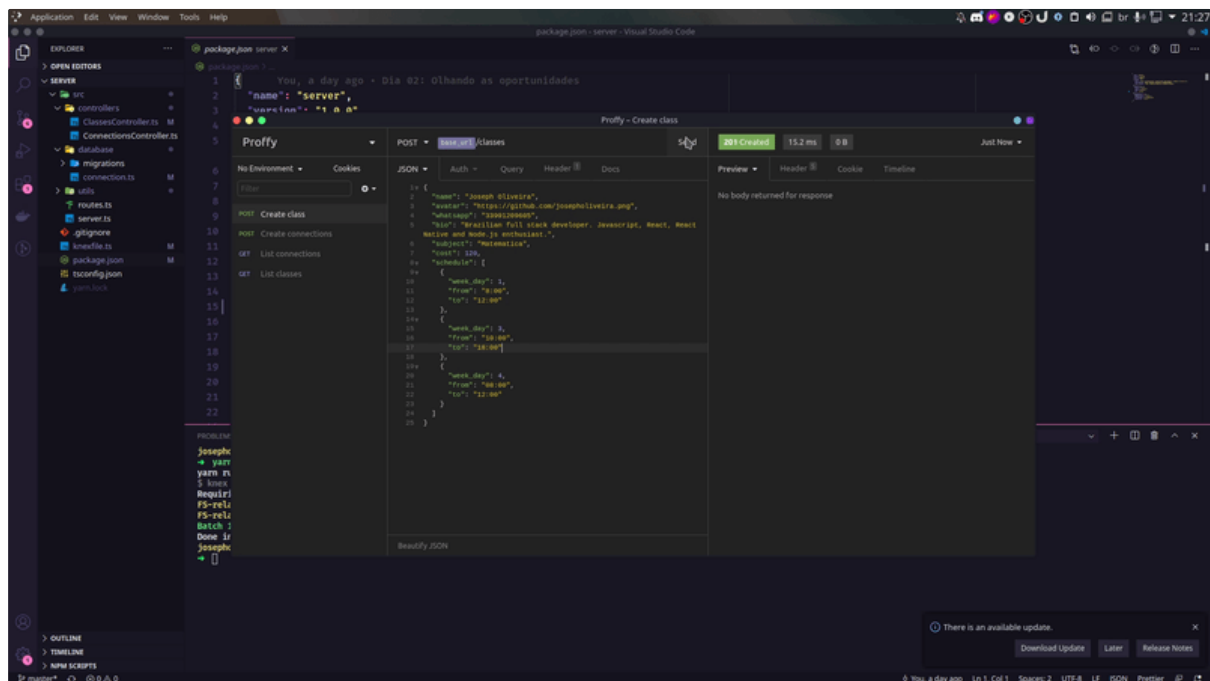
```
package.json server X
package.json > ...
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "main": "index.js",
5   "license": "MIT",
6   "scripts": {
7     "start": "tsnd --transpile-only --ignore-watch node_modules --respawn src/server.ts",
8     "knex:migrate": "knex --knexfile knexfile.ts migrate:latest",
9     "knex:rollback": "knex --knexfile knexfile.ts migrate:rollback"
10  },
11  "dependencies": {
12    "cors": "^2.8.5",
13    "express": "^4.17.1",
14    "knex": "^0.21.2",
15    "pg": "^8.3.0"
16  },
17  "devDependencies": {
18    "@types/cors": "^2.8.7",
19    "@types/express": "^4.17.7",
20    "ts-node-dev": "^1.0.0-pre.56",
21    "typescript": "^3.9.7"
22  }
}

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
josepholiveira in proffy/server on master [X!]
→ yarn knex:migrate
yarn run v1.22.4
$ knex --knexfile knexfile.ts migrate:latest
Requiring external module ts-node/register
FS-related option specified for migration configuration. This resets migrationSource to default FsMigrations
FS-related option specified for migration configuration. This resets migrationSource to default FsMigrations
Batch 1 run: 4 migrations
Done in 1.62s.
```

E olha que legal, todas as nossas migrations rodaram sem precisar fazer nenhuma alteração na estrutura das migrations! Caso você queira, pode inclusive ver o resultado no seu cliente de banco de dados:



Agora basta executar a rota de criação de classes de novo, e você verá que tudo continua funcionando perfeitamente:



Configurações avançadas

Ok! Agora que já deixamos tudo preparado, faremos penas algumas pequenas alterações tornar o nosso deploy mais seguro.

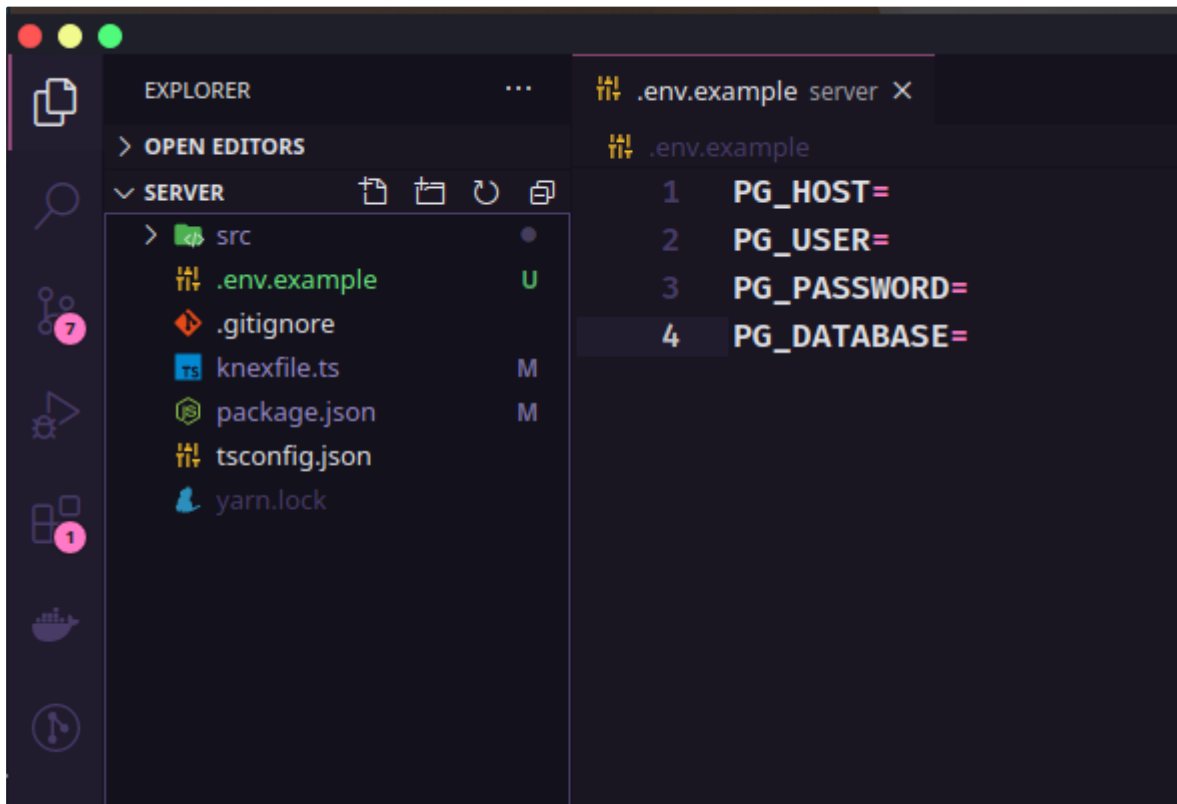
Vamos começar adicionando uma nova biblioteca na nossa aplicação, o dotenv:

```
# Com NPM npm install dotenv # Com Yarn yarn add dotenv
```

PowerShell ▾

Agora precisamos fazer algumas coisas, primeiro vamos criar dois arquivos:

O primeiro arquivo será o `.env.example` esse arquivo será responsável para guardar a estrutura que nós utilizaremos no nosso ambiente, que terão dados pessoais muitas das vezes, e nele guardaremos as informações do nosso banco de dados. Vai ficar dessa forma:

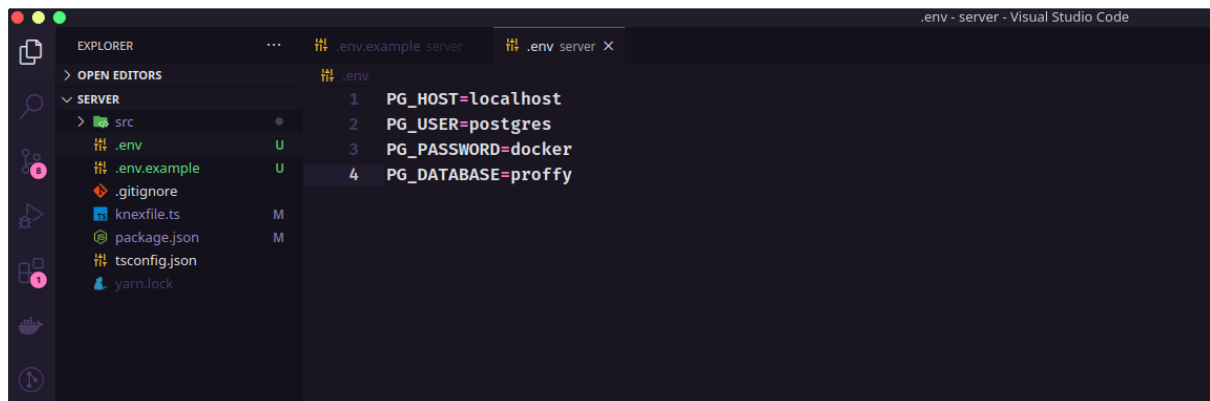


```
PG_HOST= PG_USER= PG_PASSWORD= PG_DATABASE=
```

TypeScript ▾

Agora vamos criar o segundo arquivo, o arquivo `.env`, para isso, basta copiar o seu arquivo `.env.example` e renomeá-lo para `.env`.

Esse arquivo será o que será utilizado como arquivo que conterá as configurações do ambiente local para conseguirmos configurar com os nossos dados privados do banco de dados. Então vai ficar dessa forma:



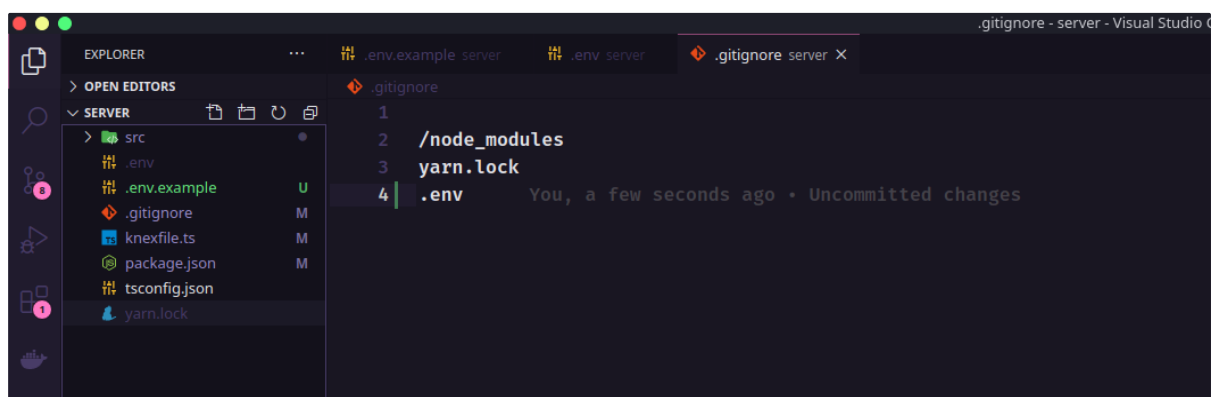
The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left displaying the project structure. The main editor window shows the `.env` file with the following content:

```
1 PG_HOST=localhost
2 PG_USER=postgres
3 PG_PASSWORD=docker
4 PG_DATABASE=proffy
```

```
PG_HOST=localhost PG_USER=postgres PG_PASSWORD=docker
PG_DATABASE=proffy
```

TypeScript ▾

E lembre-se, o arquivo `.env` contém informações muitas vezes confidenciais, então o que vamos fazer agora é adicioná-lo ao `.gitignore` para impedir que ele seja enviado ao github. Então basta criar um arquivo com nome `.gitignore` e adicionar ele lá.



The screenshot shows the Visual Studio Code interface with the Explorer sidebar on the left. The main editor window shows the `.gitignore` file with the following content:

```
1
2 /node_modules
3 yarn.lock
4 .env
```

A status bar at the bottom of the editor indicates "You, a few seconds ago • Uncommitted changes".

Agora o que resta é configurar nossa aplicação para utilizar as informações do ambiente que configuramos no nosso `.env`, então basta abrir novamente seus arquivos `knexfile.ts` e `connection.ts` e adicionar uma importação para o `dotenv/config` e alterar as credenciais para serem as que configuramos no arquivo `env`. O resultado é o seguinte:

```

1 import path from 'path'
2 import 'dotenv/config' 6K (gzipped: 2.5K)
3
4 module.exports = {
5   client: 'pg',
6   connection: {
7     host : process.env.PG_HOST,
8     user : process.env.PG_USER,
9     password : process.env.PG_PASSWORD,
10    database : process.env.PG_DATABASE,
11  },
12  migrations: {
13    directory: path.resolve(__dirname, 'src', 'database', 'migrations')
14  },
15  useNullAsDefault: true
16 }

```

knexfile.ts

Caso você queira copiar a knexfile vai ficar assim:

```

import path from 'path' import 'dotenv/config' module.exports = {
  client: 'pg', connection: { host : process.env.PG_HOST, user :
  process.env.PG_USER, password : process.env.PG_PASSWORD, database :
  process.env.PG_DATABASE, }, migrations: { directory:
  path.resolve(__dirname, 'src', 'database', 'migrations') },
  useNullAsDefault: true }

```

TypeScript ▾

```

1 import knex from 'knex'
2 import 'dotenv/config' 6K (gzipped: 2.5K)
3
4 const db = knex({
5   client: 'pg',
6   connection: {
7     host : process.env.PG_HOST,
8     user : process.env.PG_USER,
9     password : process.env.PG_PASSWORD,
10    database : process.env.PG_DATABASE,
11  },
12  useNullAsDefault: true
13 })
14
15 export default db

```

connection.ts

Caso você queira copiar a connection vai ficar assim:

```
import knex from 'knex' import 'dotenv/config' const db = knex({
  client: 'pg', connection: { host : process.env.PG_HOST, user :
  process.env.PG_USER, password : process.env.PG_PASSWORD, database :
  process.env.PG_DATABASE, }, useNullAsDefault: true }) export default
db
```

TypeScript ▾

Prontinho, agora basta reiniciar o seu server e rodar o `yarn start` novamente e você vai ver que tudo continua funcionando perfeitamente!

Agora precisamos fazer algumas alterações nos nossos scripts no package.json, eu deixei os meus dessa forma:

```
"scripts": {
  "start": "node dist/src/server.js",
  "dev": "tsnd --transpile-only --ignore-watch node_modules --respawn src/server.ts",
  "knex:migrate": "knex --knexfile dist/knexfile.js migrate:latest",
  "knex:rollback": "knex --knexfile dist/knexfile.js migrate:rollback",
  "build": "tsc"
},
```

Lembre-se de trocar a knexfile para o caminho da knexfile.js conforme no print

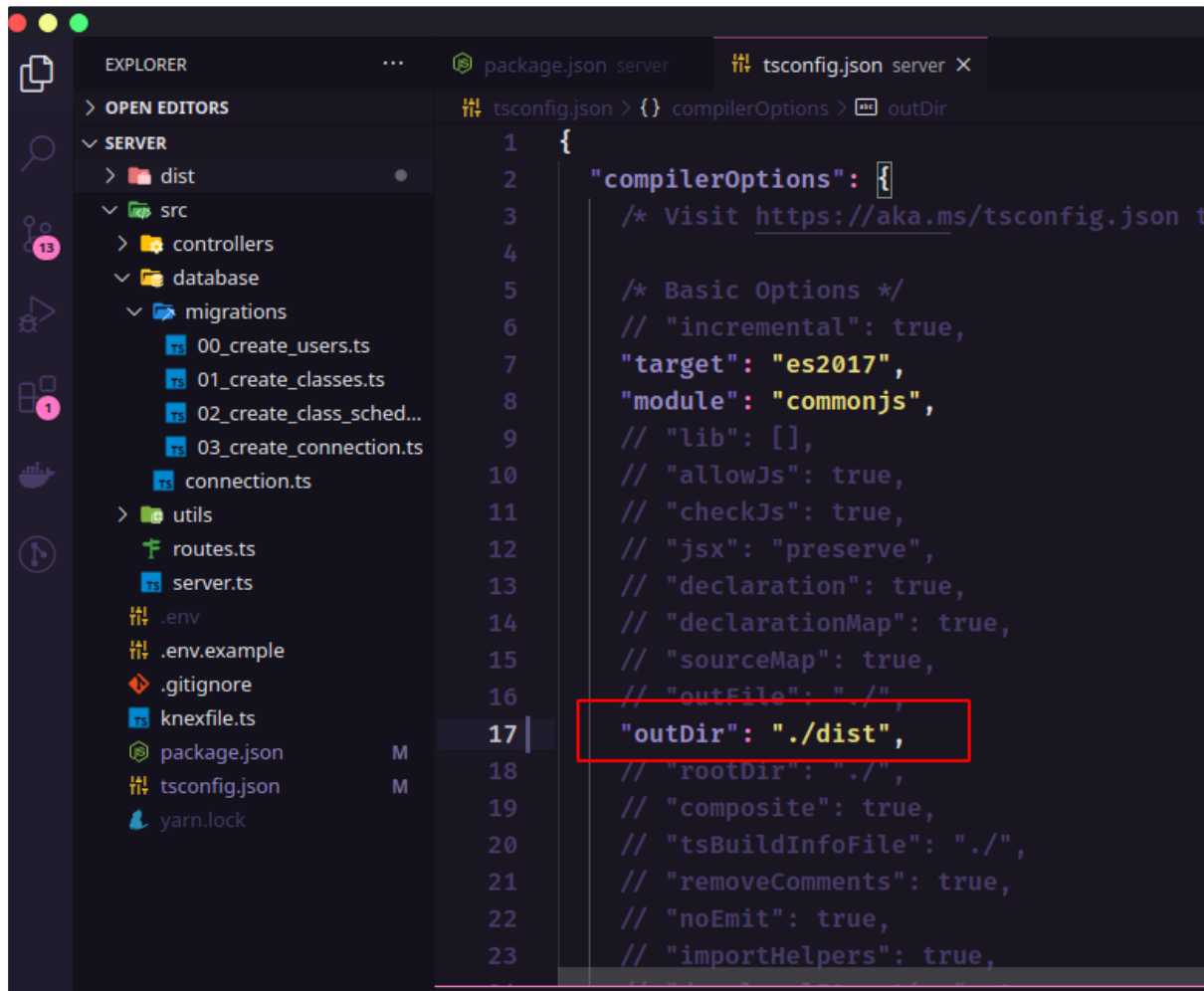
Caso você queira copiar os scripts, vai ficar assim:

```
"scripts": { "start": "node dist/src/server.js", "dev": "tsnd --
transpile-only --ignore-watch node_modules --respawn src/server.ts",
"knex:migrate": "knex --knexfile dist/knexfile.js migrate:latest",
"knex:rollback": "knex --knexfile dist/knexfile.js
migrate:rollback", "build": "tsc" },
```

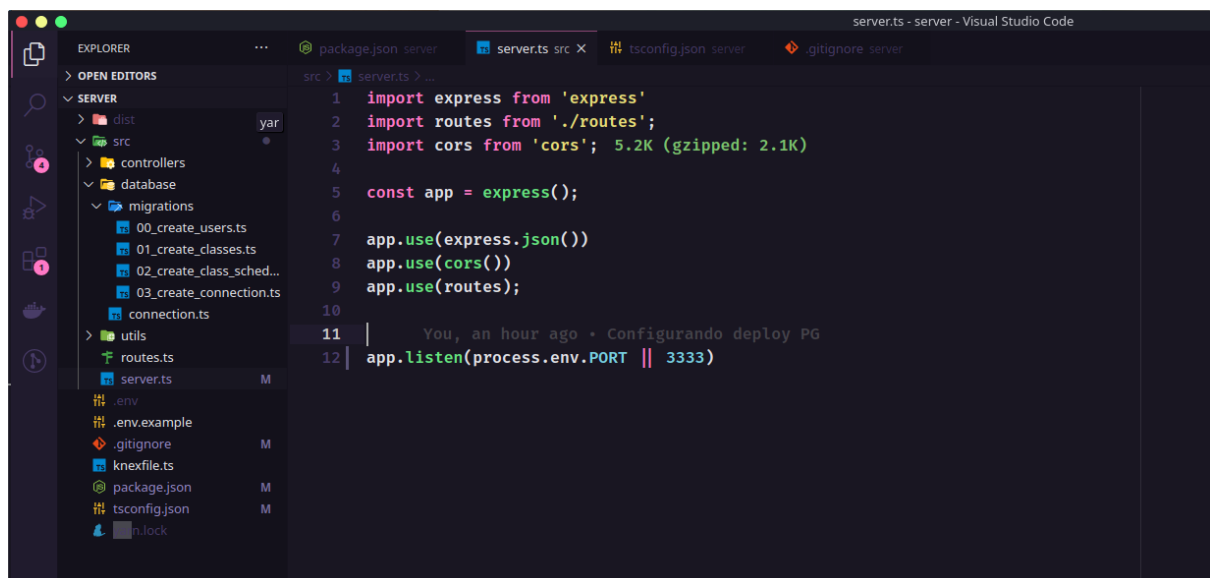
TypeScript ▾

PS: Com essa alteração, você deverá passar a rodar `yarn dev` e não `yarn start` para executar seu servidor localmente caso queira continuar testando na sua máquina.

Agora como últimos ajustes, vamos configurar o nosso tsconfig.json para gerar o build na pasta dist:



Agora como última alteração, vamos no nosso server.ts e adicionamos a `process.env.PORT` ao nosso `app.listen()`. Isso irá garantir que o Heroku consiga injetar uma porta para iniciar a aplicação.



```
app.listen(process.env.PORT || 3333)
```

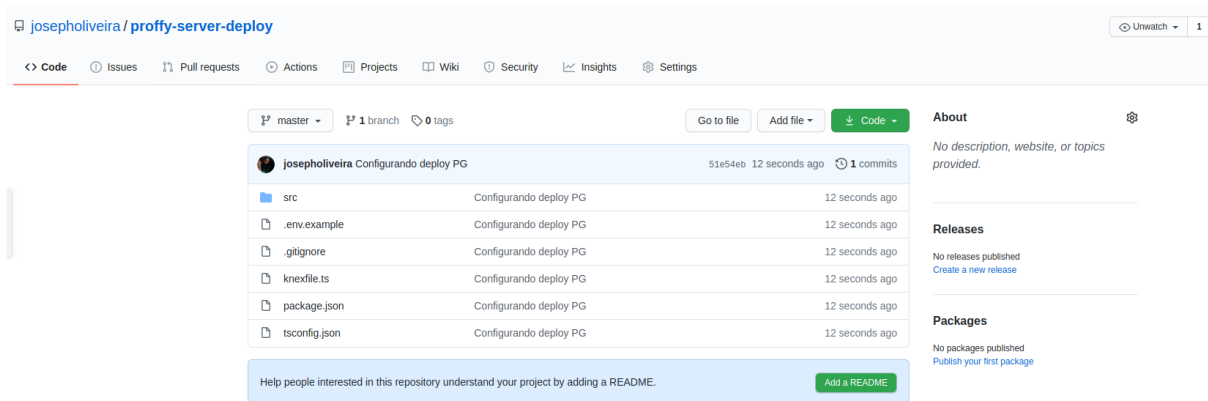
TypeScript ▾

Segundo passo - Enviando nossa aplicação ao GitHub

Claro, após fazer nossas alterações, nós apenas precisamos enviar ela ao GitHub. No meu caso eu já tinha ela no GitHub então não precisei fazer nada.

Agora, para garantir que não teremos erro nenhum, vamos criar um repositório contendo apenas os arquivos do nosso server no github, para evitar qualquer tipo de conflito entre pastas. Você pode ver meu repositório aqui:

<https://github.com/josepholiveira/proffy-server-deploy>



Caso você não saiba como utilizar o GitHub, recomendo ver esse vídeo para entender tudo que você precisa para iniciar:

Como usar Git e Github na prática: Guia para iniciantes | Mayk B...



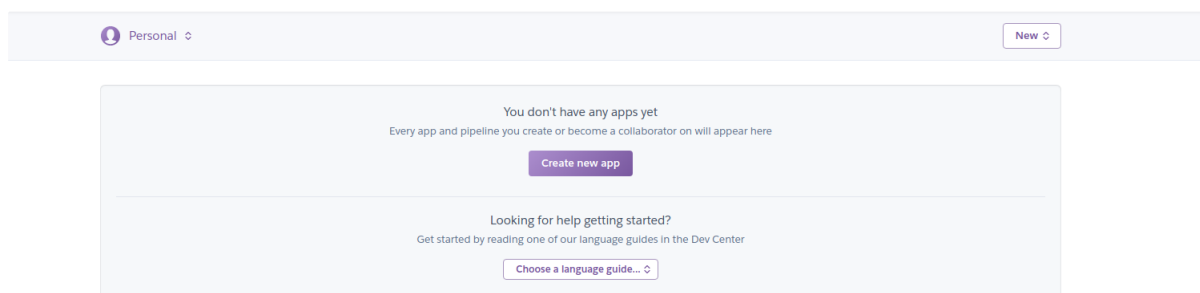
Terceiro passo - Enviando a aplicação ao Heroku

Finalmente a parte que você tanto esperava, o **deploy**! Agora que estamos 100% preparados com nossa aplicação, com banco de dados PostgreSQL e só esperando para ser colocada em produção, vamos começar acessando a página do [Heroku](#), para isso, [clique aqui](#).

Preparando o deploy

Estando na página inicial do Heroku, no canto superior direito existem duas opções: Log In e Sign Up, para você logar ou se cadastrar. Basta clicar em uma delas e seguir os passos. Caso você já tenha uma conta basta logar nela.

Feito isso, você deverá ter a seguinte tela:



Apenas clique no botão "Create New App" e você terá a seguinte tela, onde eu dei um nome para o meu App (o nome deve ser único):

The screenshot shows the 'Create New App' form in Heroku. It has a section for 'App name' with a text input field containing 'proffy-deploy' and a green checkmark icon to its right. Below the input field, it says 'proffy-deploy is available'. The next section is 'Choose a region' with a dropdown menu showing 'United States' and a flag icon. Below this is a link 'Add to pipeline...' with a plus icon. At the bottom is a purple button labeled 'Create app'.

Agora clique em **Create app** e você vai ser redirecionado para essa tela, onde vamos selecionar "Connect to Github":

Overview Resources **Deploy** Metrics Activity Access Settings

Add this app to a pipeline
Create a new pipeline or choose an existing one and add this app to a stage in it.

Add this app to a stage in a pipeline to enable additional features
Pipelines let you connect multiple apps together and **promote code** between them. [Learn more](#)
Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. [Learn more](#)

Choose a pipeline

Deployment method

Heroku Git Use Heroku CLI

GitHub Connect to GitHub

Container Registry Use Heroku CLI

Connect to GitHub
Connect this app to GitHub to enable code diffs and deploys.

View your code diffs on GitHub
Connect your app to a GitHub repository to see commit diffs in the activity log.

Deploy changes with GitHub
Connecting to a repository will allow you to deploy a branch to your app.

Automatic deploys from GitHub
Select a branch to deploy automatically whenever it is pushed to.

Create review apps in pipelines
Pipelines connected to GitHub can enable **review apps**, and create apps for new pull requests. [Learn more](#)

Connect to GitHub

Após clicar nos dois botões, ele irá abrir um popup pedindo permissão para se conectar ao seu github, você apenas precisa permitir a conexão e seguimos pro próximo passo.

Agora basta escrever o nome do seu repositório criado no github, e clicar em search.

Connect to GitHub
Connect this app to GitHub to enable code diffs and deploys.

Search for a repository to connect to

josepholiveira proffy-server-deploy Search

Missing a GitHub organization? [Ensure Heroku Dashboard has team access.](#)

Após encontrar o repositório, basta clicar em connect.

Connect to GitHub
Connect this app to GitHub to enable code diffs and deploys.

Search for a repository to connect to

josepholiveira proffy-server-deploy Search

Missing a GitHub organization? [Ensure Heroku Dashboard has team access.](#)

josepholiveira/proffy-server-deploy Connect

Agora para configurar o buildpack que irá fazer deploy da aplicação, vá até a **aba settings** da página do seu app.

Agora vamos descer um pouco a página e vamos adicionar alguns buildpacks, e lá clicamos no botão Add Buildpack e a seguinte opção no modal que vai aparecer:

- heroku/nodejs

Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

[Reveal Config Vars](#)

Buildpacks

Buildpacks are scripts that are run when your app is deployed. They are used to install dependencies for your app and configure your environment. [Find new buildpacks on Heroku Elements](#)

Your new buildpack configuration will be used when this app is next deployed.

[Add buildpack](#)

Add Buildpack

Enter Buildpack URL

heroku/nodejs

Or select from our officially supported buildpacks

nodejs

python

php

ruby

java

go

gradle

scala

clojure

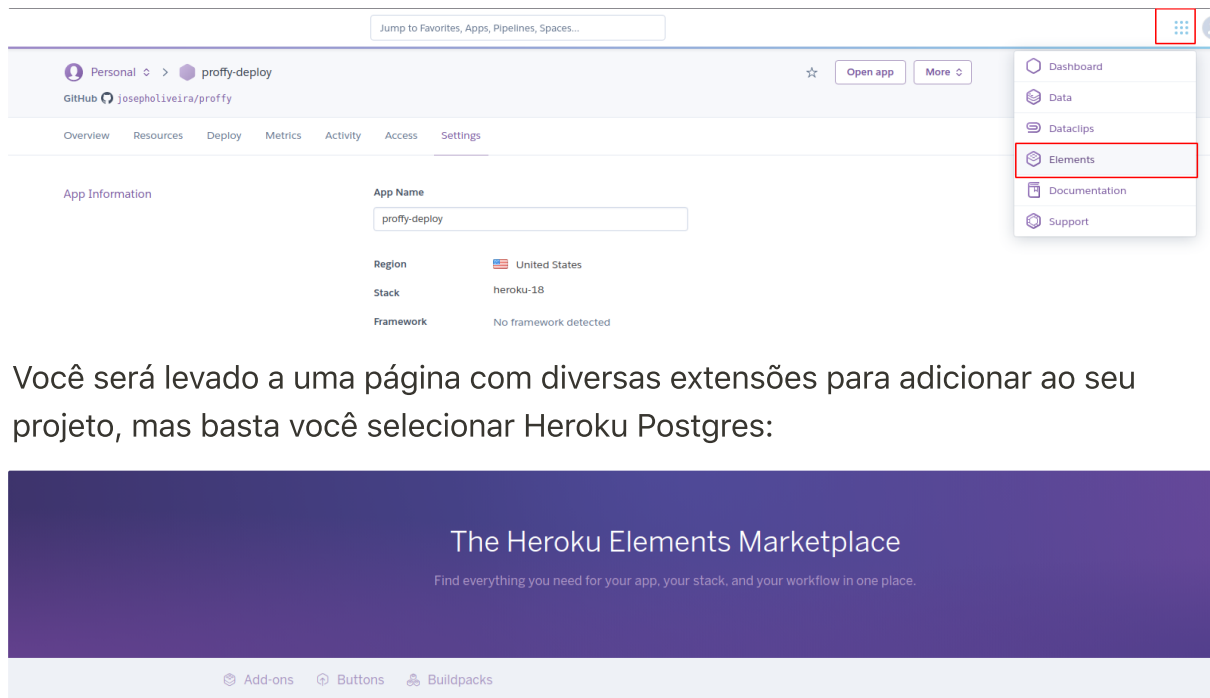
Save changes

Quarto passo - Preparando o banco de dados

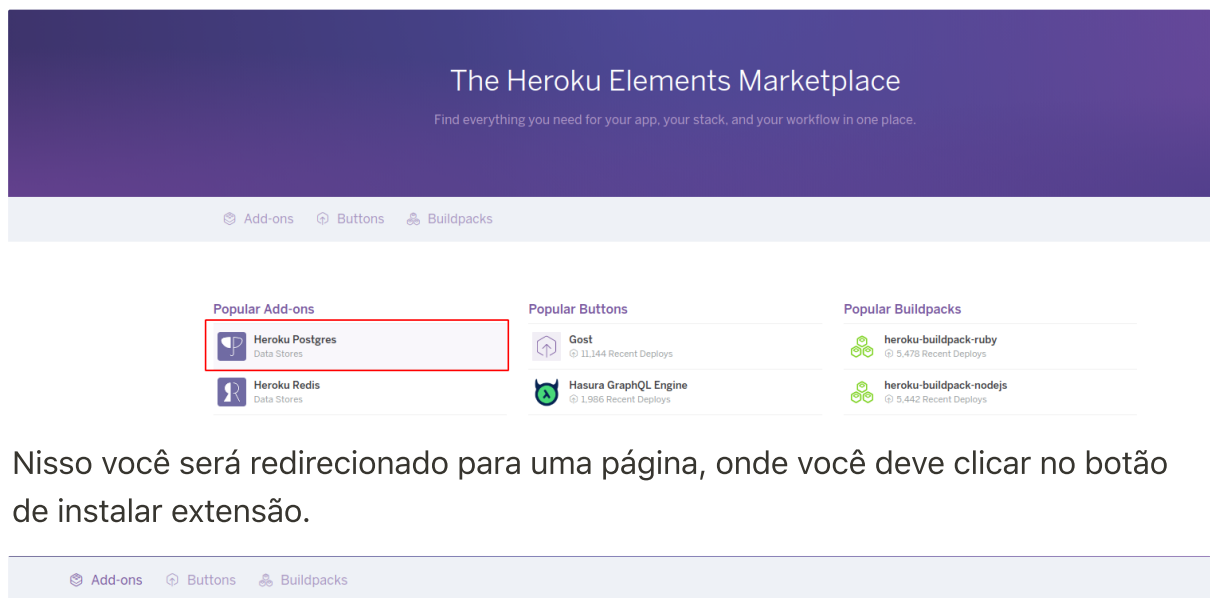
Bacana, por enquanto tudo certo! Agora vamos configurar nosso banco de dados.

Criando a instância do banco de dados

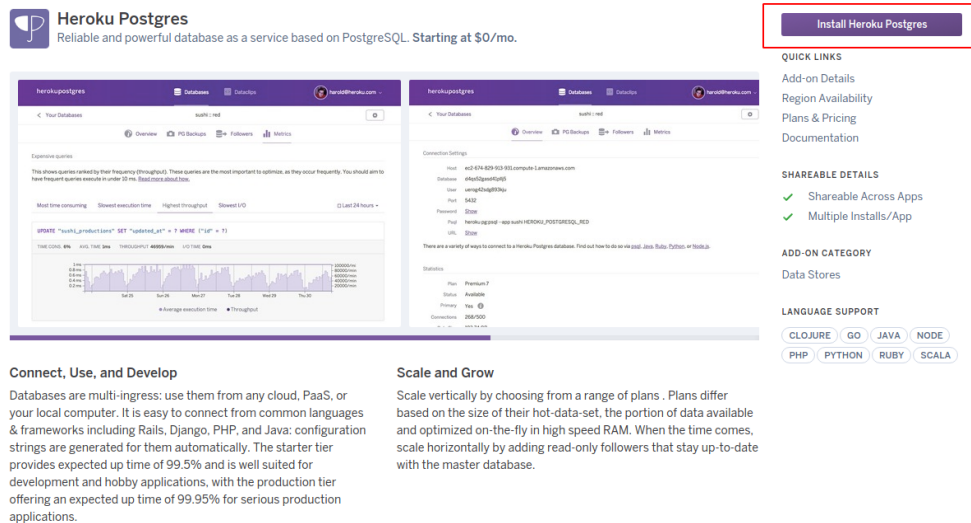
Primeiramente, basta acessar o menu no canto superior direito e depois clicar em Elements.



Você será levado a uma página com diversas extensões para adicionar ao seu projeto, mas basta você selecionar Heroku Postgres:




Nisso você será redirecionado para uma página, onde você deve clicar no botão de instalar extensão.



Region Availability

Agora é só selecionar o app que você acabou de criar, clicar no botão Provision add-on e pronto!

Provision add-on

 Provision this add-on to an app
Heroku Postgres
[View on the Elements Marketplace](#)

Add-on plan

Hobby Dev - Free

App to provision to

pro

proffy-deploy

By provisioning this add-on, I agree to the [Terms of Service](#)

Provision add-on

Agora na página que você foi redirecionado, clique no nome da extensão instalada:

Personal > proffy-deploy

GitHub josepholiveira/proffy

Overview Resources Deploy Metrics Activity Access Settings


Dynos

This app has no process types yet
Add a Profile to your app in order to define its process types. [Learn more](#)

Add-ons

Find more add-ons

Quickly add add-ons from Elements

 Heroku Postgres Attached as DATABASE Hobby Dev Free

Estimated Monthly Cost \$0.00

Na nova página, clique em Settings e depois em View Credentials:

Overview Durability Settings Dataclips

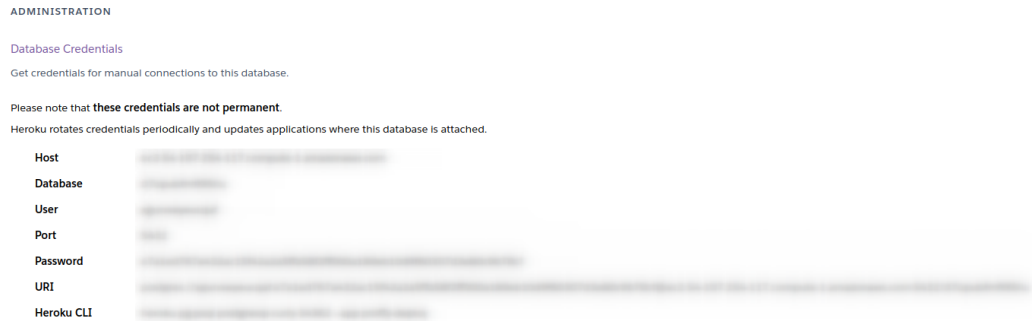
ADMINISTRATION

Database Credentials
Get credentials for manual connections to this database. [View Credentials...](#)

Reset Database
Reset the database to its originally-provisioned state, deleting all data inside it. [Reset Database...](#)

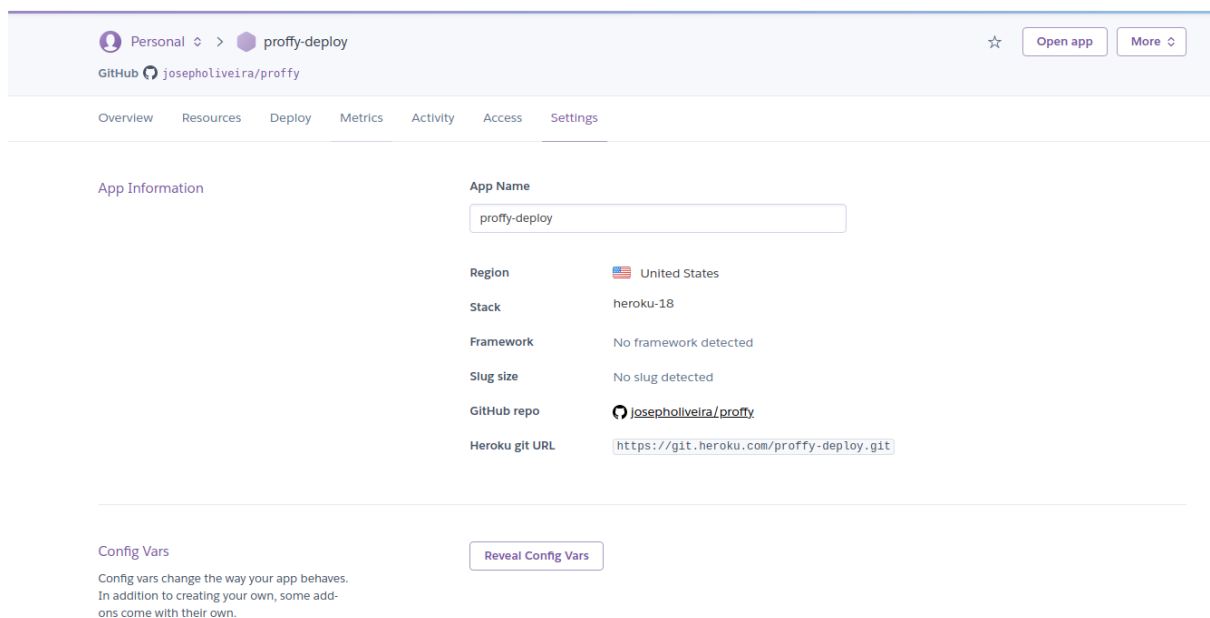
Destroy Database
Destroys the database and all of the data inside it. [Destroy Database...](#)

Aqui você terá todos os dados que você irá utilizar para conectar-se ao banco de dados, então anote eles que vamos utilizar em breve:

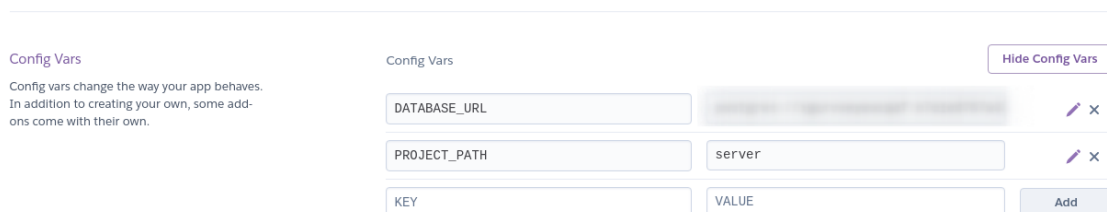


Configurando credenciais do banco de dados

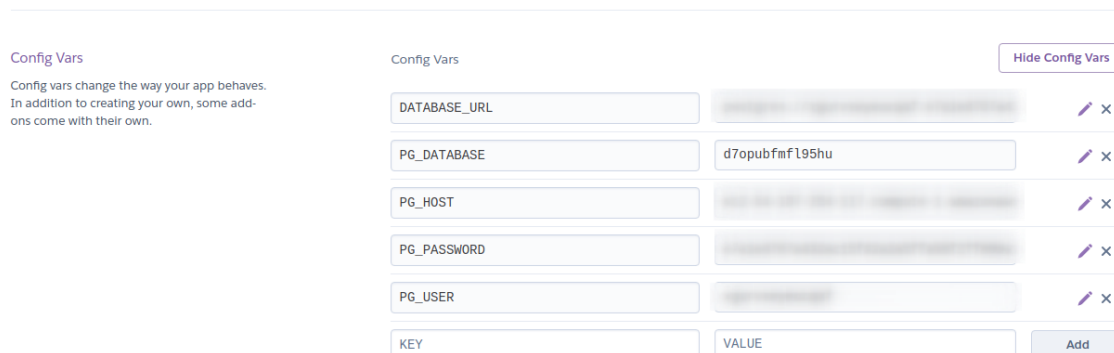
Agora com os dados do seu banco de dados, você precisa voltar a página do seu App na sua dashboard do heroku, clicar em settings novamente e ir até a parte onde tem "Config Vars"



Você verá que o heroku automaticamente vai adicionar um DATABASE_URL mas não vamos utilizar ele, vamos configurar nossas próprias variáveis:



Para isso, adicione exatamente as mesmas variáveis que criamos no nosso arquivo .env do banco de dados, e preencha elas com os valores que você anotou da sua instância do Postgres Heroku.



The screenshot shows the Heroku Config Vars management page. On the left, there's a section titled 'Config Vars' with a description: 'Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.' On the right, there's a table of existing config vars. The table has two columns: 'KEY' and 'VALUE'. The rows are: DATABASE_URL (value is a long alphanumeric string), PG_DATABASE (value is d70pubbfmfl95hu), PG_HOST (value is a long alphanumeric string), PG_PASSWORD (value is a long alphanumeric string), and PG_USER (value is a long alphanumeric string). Each row has edit and delete icons. At the bottom, there are input fields for 'KEY' and 'VALUE', and an 'Add' button.

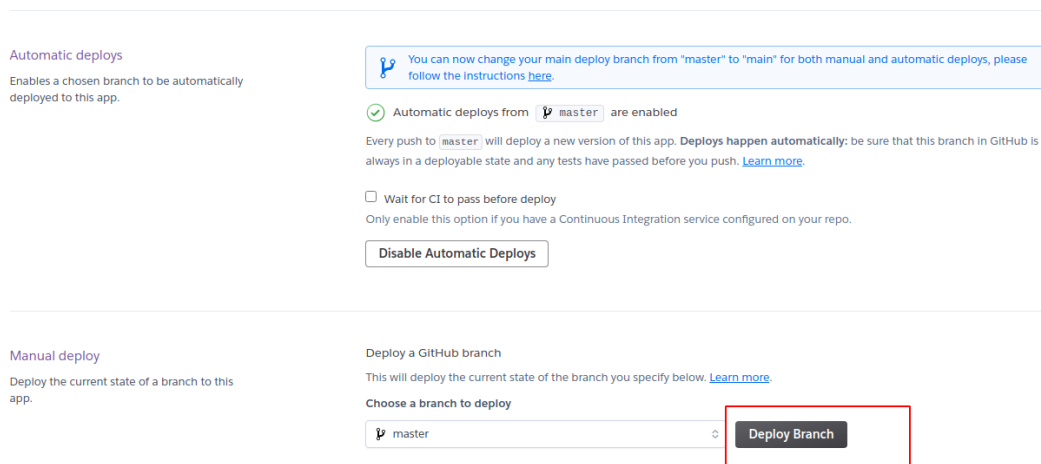
KEY	VALUE	
DATABASE_URL	postgres://[redacted]:[redacted]@[redacted].herokuapp.com/[redacted]	✎ ✕
PG_DATABASE	d70pubbfmfl95hu	✎ ✕
PG_HOST	[redacted]	✎ ✕
PG_PASSWORD	[redacted]	✎ ✕
PG_USER	[redacted]	✎ ✕
KEY	VALUE	Add

PS: Veja que na DATABASE eu utilizei a que o heroku gerou pra mim, para agilizar o processo.

Finalizando o deploy do servidor

Agora o momento mais emocionante, o deploy!

Voltamos na nossa Dashboard do app que criamos no heroku, clicamos no menu Deploy e selecionamos a seguinte opção:



The screenshot shows the Heroku Deploy interface. At the top, there's a message: 'You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions [here](#).' Below this, there's a section for 'Automatic deploys' with a green checkmark and the text: 'Automatic deploys from master are enabled'. It explains that every push to master will deploy a new version of the app. There's a checkbox for 'Wait for CI to pass before deploy' which is currently unchecked. Below this is a 'Disable Automatic Deploys' button. At the bottom, there's a section for 'Manual deploy' with the text: 'Deploy the current state of a branch to this app.' Below this is a 'Deploy a GitHub branch' section with the text: 'This will deploy the current state of the branch you specify below. [Learn more](#).' There's a dropdown menu for 'Choose a branch to deploy' with 'master' selected. To the right of the dropdown is a 'Deploy Branch' button, which is highlighted with a red rectangle.

Para o nosso servidor é apenas isso, agora basta rodar as nossas migrations!

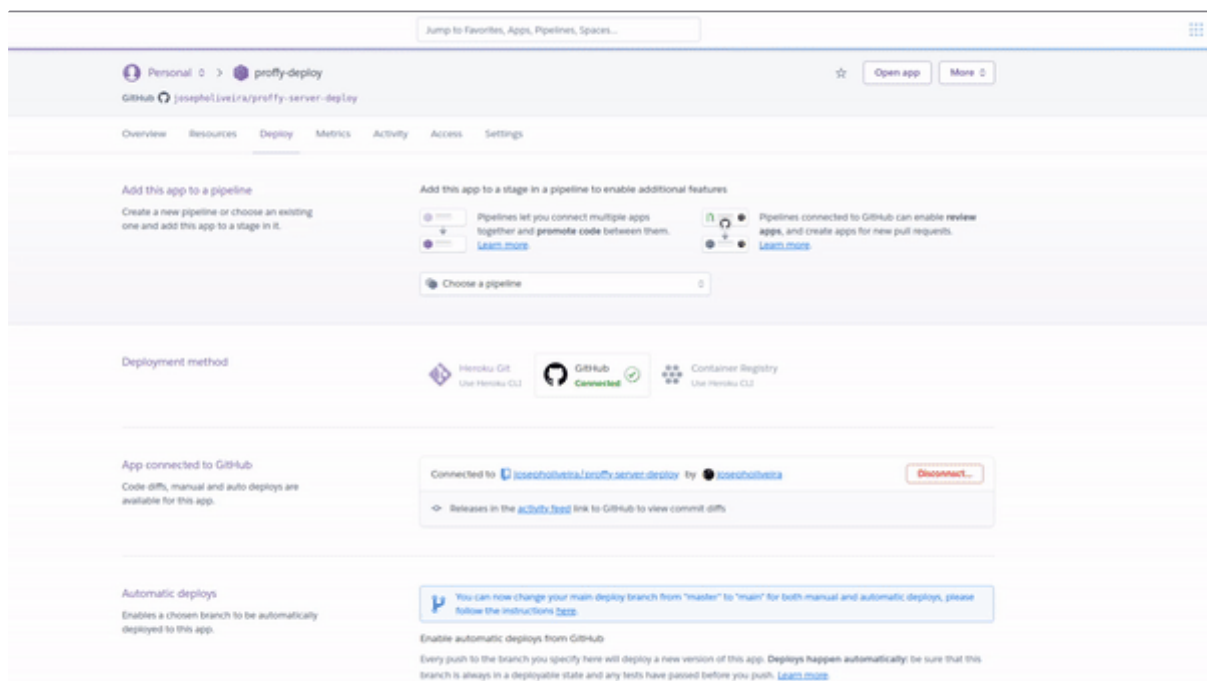
Executando as migrations

Para executar as migrations é simples, basta clicar em "More" na sua dashboard, depois clicar em "Run Console", depois digitar "bash" no campo de texto que aparece dentro do modal e executar e clicar no botão "run".

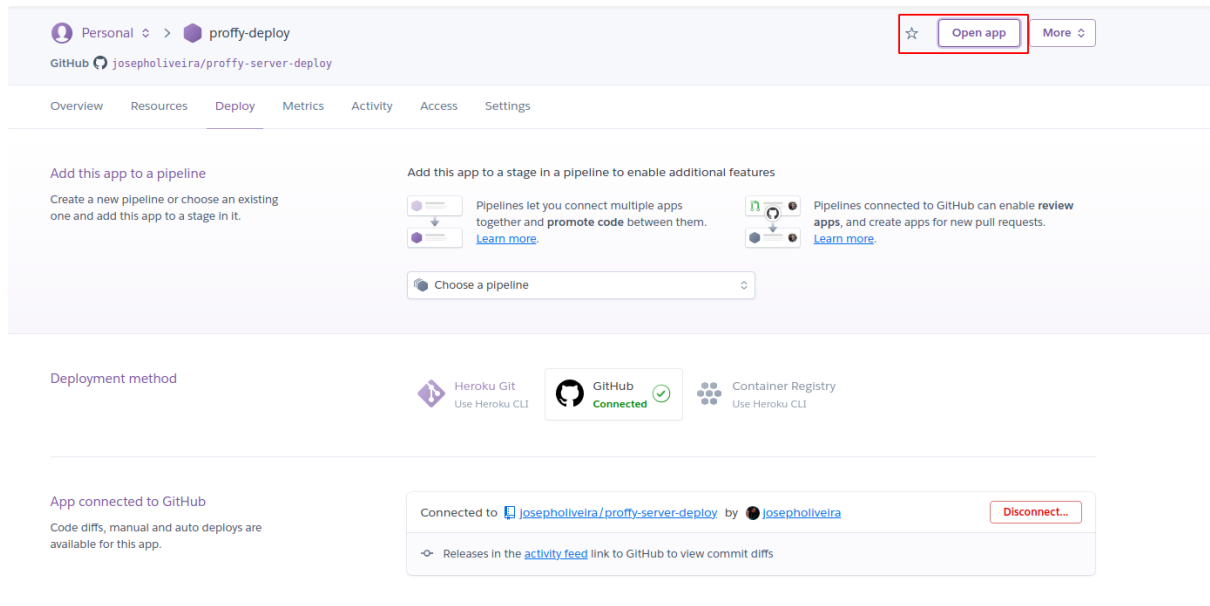
Após isso basta executar o comando para rodar as migrations com o npm:

```
npm run knex:migrate
```

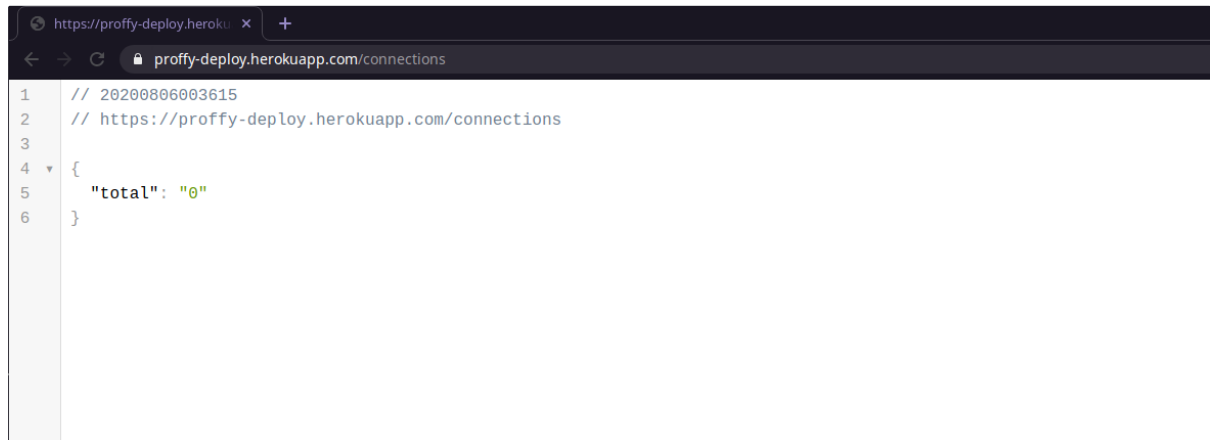
Shell ▾



Prontinho! Suas migrations foram executadas! Agora você poderá acessar seu servidor que você acabou de realizar o deploy pela URL que você escolheu ao criar o projeto ou clicando no botão "Open app":



Agora é só acessar uma rota válida, e prontinho, já vai estar na tela!



```
1 // 20200806003615
2 // https://proffy-deploy.herokuapp.com/connections
3
4 {
5   "total": 0
6 }
```

É isso aí, terminamos o deploy do nosso servidor, parabéns por ter chego até aqui!



Deploy do Front-end Web da aplicação

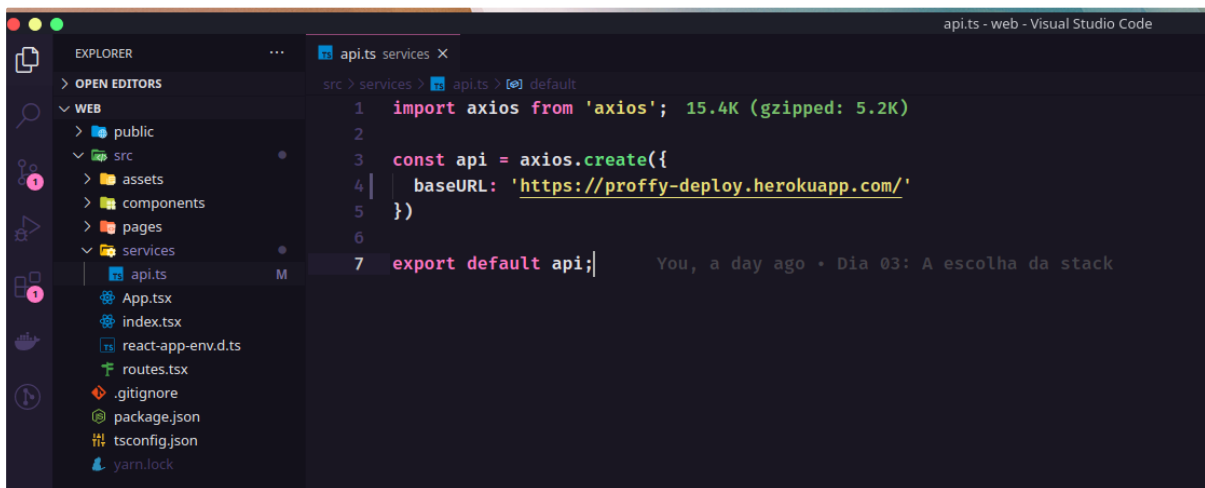
Tenho certeza que você esperava por esse momento! Agora nós vamos enviar nosso frontend web para poder ser acessado por toda a internet!

Para isso vamos seguir alguns passos que nós já seguimos anteriormente, como preparar o nosso frontend.

Primeiro passo - Conectando a API

Certo, vamos iniciar do início! Para isso, na sua máquina, abra a pasta que você criou o seu front-end web criado com React no seu VSCode. No meu caso o nome dela é `web`.

A primeira coisa que vamos fazer é abrir o arquivo `api.ts` para configurar a conexão com nossa API, e vamos trocar a baseURL do nosso `localhost` para a URL que o heroku nos deu. No meu caso vai ficar como no print, mas lembre-se de utilizar a sua própria URL do backend que fizemos deploy logo acima.



```
import axios from 'axios'; const api = axios.create({ baseURL:
'URL_DO_SEU_SERVIDOR' }) export default api;
```

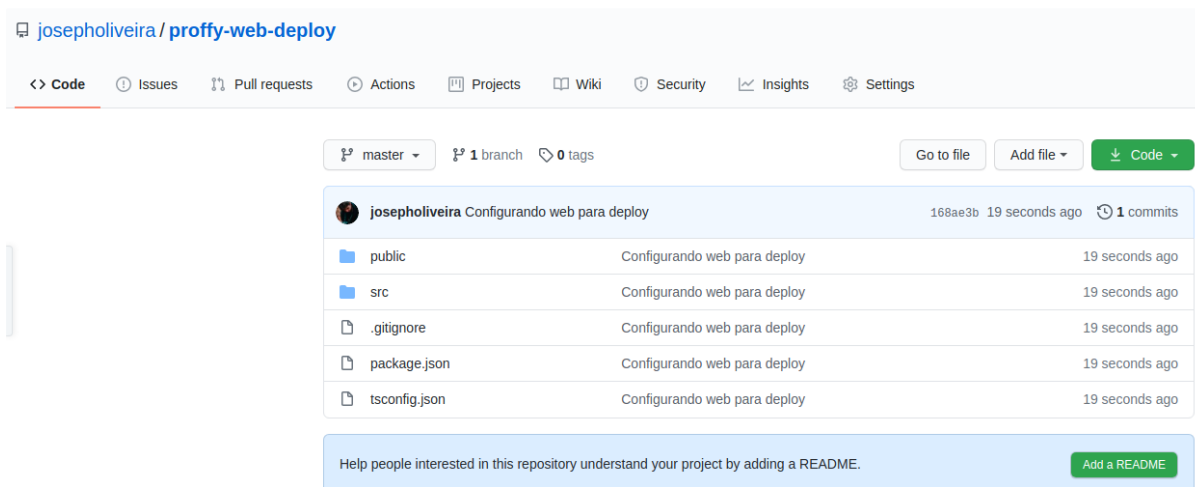
TypeScript ▾

Segundo passo - Enviando nossa aplicação ao GitHub

Claro, após fazer nossas alterações, nós apenas precisamos enviar ela ao GitHub.

Agora, para garantir que não teremos erro nenhum, vamos criar um repositório contendo apenas os arquivos do nosso server no github, para evitar qualquer tipo de conflito entre pastas. Você pode ver meu repositório aqui:

<https://github.com/josepholiveira/proffy-web-deploy>



Caso você não saiba como utilizar o GitHub, recomendo ver esse vídeo para entender tudo que você precisa para iniciar:

Como usar Git e Github na prática: Guia para iniciantes | Mayk B...



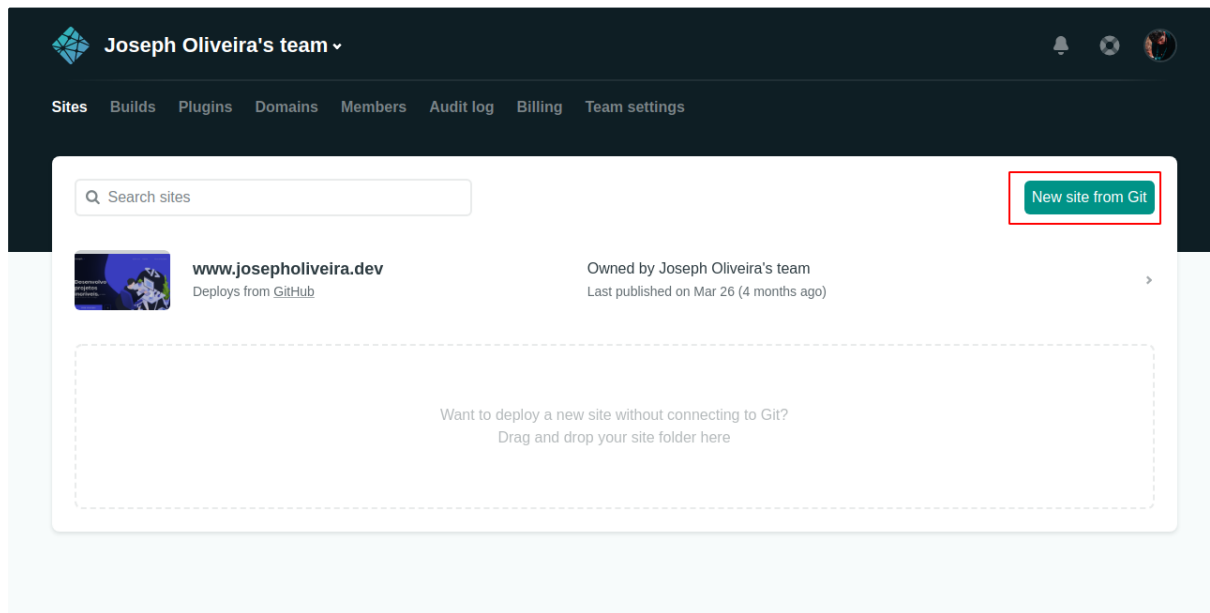
Terceiro passo - Enviando a aplicação ao Netlify

Finalmente a parte que você tanto esperava, o **deploy**! Agora que estamos 100% preparados com nossa aplicação, com banco de dados PostgreSQL e só esperando para ser colocada em produção, vamos começar acessando a página do [Netlify](#), para isso, [clique aqui](#).

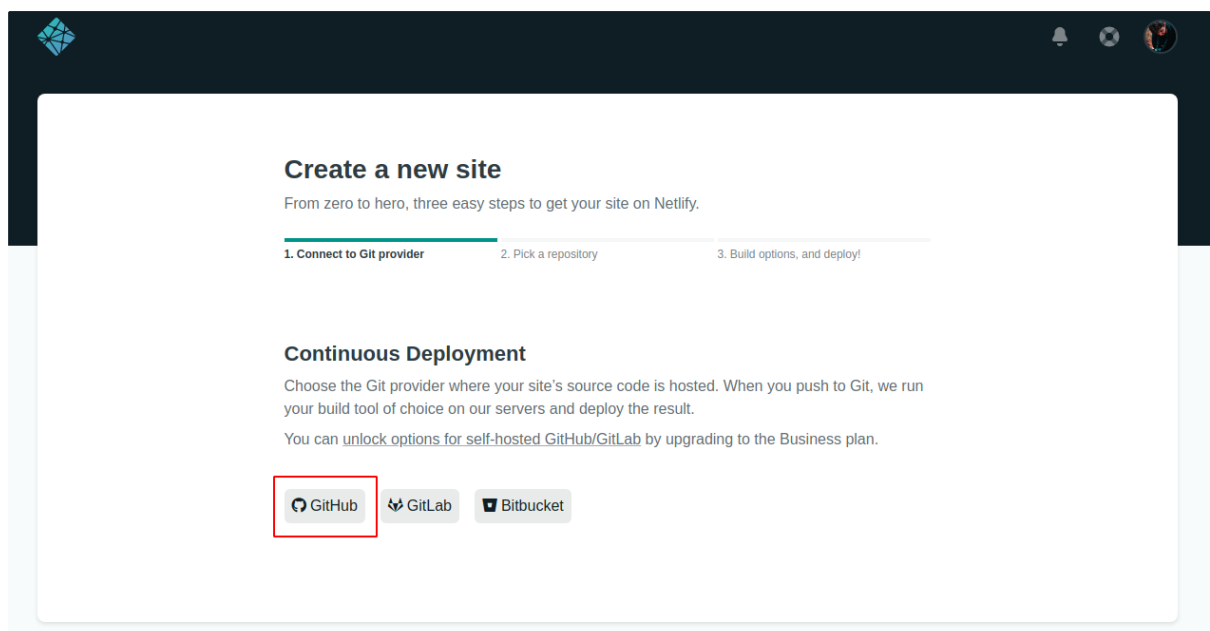
No canto superior direito da página do Netlify, você possui as opções de se cadastrar ou logar, eu pessoalmente me logo com o GitHub para acelerar o processo, então crie sua conta ou logue em uma já existente e eu te espero na Dashboard do netlify!

Criando uma nova aplicação

Agora na sua dashboard do Netlify, crie uma nova aplicação criando no botão no canto superior direito:



Com seu código no github, basta você clicar no botão Github que irá aparecer:



Nessa hora, deve aparecer um PopUp pedindo autorização para se conectar ao seu Github, permita que ele se conecte. Caso já tenha conectado, apenas aguarde alguns segundos que ele irá te redirecionar para a página onde você vai selecionar o repositório com seus arquivos da versão web do Proffy. No meu caso o nome do repositório que eu criei foi `proffy-web-deploy`, mas no seu caso, procure pelo nome do repositório que você criou.

Create a new site

From zero to hero, three easy steps to get your site on Netlify.


1. Connect to Git provider


2. Pick a repository

3. Build options, and deploy!

Continuous Deployment: GitHub App


Choose the repository you want to link to your site on Netlify. When you push to Git, we run your build tool of choice on our servers and deploy the result.

 josepholiveira ▾

 proffy

 josepholiveira/proffy >

 josepholiveira/proffy-server-deploy >

 josepholiveira/proffy-web-deploy >

Can't see your repo here? [Configure the Netlify app on GitHub.](#)

Agora basta clicar em Deploy Site, e aguardar o processo de deploy da aplicação:

Create a new site

From zero to hero, three easy steps to get your site on Netlify.

1. Connect to Git provider

2. Pick a repository

3. Build options, and deploy!

Deploy settings for josepholiveira/proffy-web-deploy

Get more control over how Netlify builds and deploys your site with these settings.

Owner

Joseph Oliveira's team

Branch to deploy

master

Basic build settings

If you're using a static site generator or build tool, we'll need these settings to build your site.

[Learn more in the docs](#)

Build command

npm run build

Publish directory

build/

Show advanced

Deploy site

Finalizando o Deploy

Depois de ter feito o Deploy do seu site, você pode acompanhar o status dele pela seguinte tela na Dashboard do seu app:

musing-kare-9f9f21

- Site deploy in progress

Deploys from GitHub. Created at 12:09 PM.

Site settings Domain settings

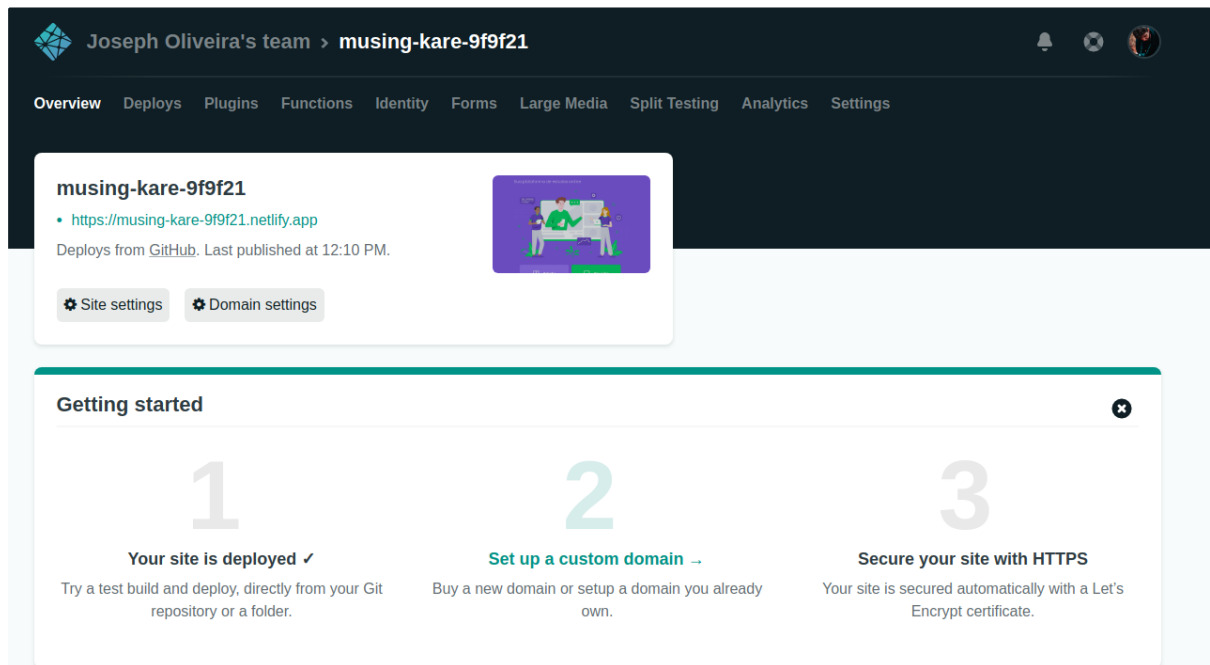
Getting started

1Deploying your siteNetlify's robots are busy building and deploying your site to our CDN.

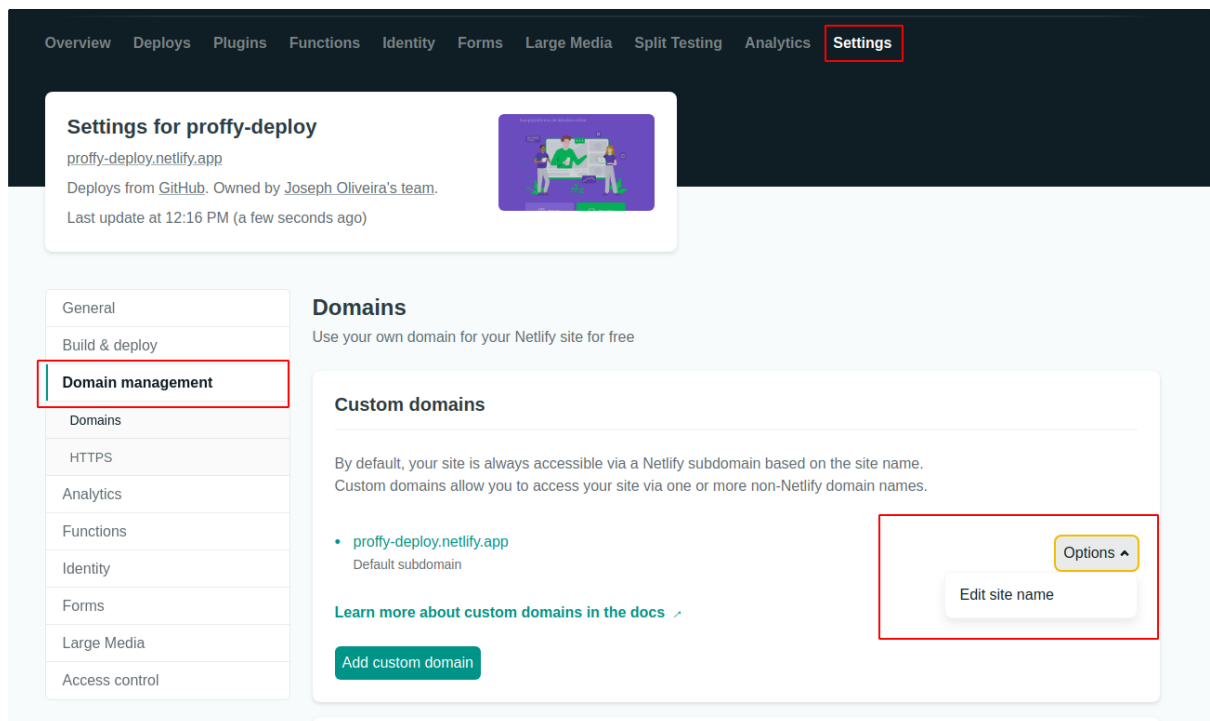
2Set up a custom domainBuy a new domain or setup a domain you already own.

3Secure your site with HTTPSYour site is secured automatically with a Let's Encrypt certificate.

Após aguardar o deploy se concluir, ele irá te dar o link do deploy que foi gerado:



Dica: Caso você queira alterar a URL do seu deploy, basta você ir nas configurações do seu app e alterar o nome do domínio dele aqui:



Agora é só acessar esse link e ver sua aplicação funcionando, parabéns!! 🎉

