# Persistência de dados com PostgreSQL e ASP.NET Core no MacOS

[Facebook](https://www.facebook.com/sharer/sharer.php?u=http://www.devmedia.com.br/persistencia-de-dados-com-postgresql-e-asp-net-core-no-macos/36900" \o "Compartilhe no Facebook" \t "_blank) [Twitter](https://twitter.com/home?status=Persist%EAncia%20de%20dados%20com%20PostgreSQL%20e%20ASP.NET%20Core%20no%20MacOS%0D%0Ahttp://www.devmedia.com.br/persistencia-de-dados-com-postgresql-e-asp-net-core-no-macos/36900" \o "Compartilhe no Twitter" \t "_blank)

 (0)  (0)

## Neste artigo veremos como configurar e utilizar o PostgreSQL para persistência de dados em um projeto ASP.NET Core com Identity e Entity Framework.

**Motivação**

A possibilidade de [desenvolver para o .NET](http://www.devmedia.com.br/cursos/net) a partir de qualquer sistema operacional abriu um leque de possibilidades no que se refere ao direito de escolha do desenvolvedor, incluindo, por exemplo, o banco de dados. Por isso, é importante entender como conectar uma aplicação corretamente com outros servidores de banco de dados diferentes do SQL Server e SQLite, opções padrão em projetos ASP.NET Core para Windows e MacOS/Linux, respectivamente.

Nesse artigo, iremos apresentar, de maneira introdutória, como instalar o banco de dados PostgreSQL em um ambiente MacOS e conectá-lo com uma aplicação ASP.NET Core, desenvolvida no decorrer desse estudo.

### Passo 1: Instalação do PostgreSQL

Antes de mais nada, devemos instalar o servidor do PostgreSQL localmente para que seja possível o desenvolvimento e a realização de testes na aplicação. Para tanto, no terminal, digite o seguinte comando:

|  |
| --- |
| ~$ brew update  ~$ brew install postgresql |

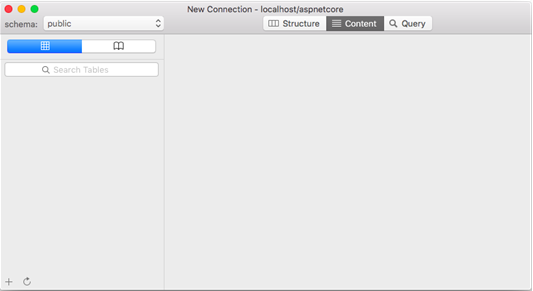
Agora, vamos criar um banco de dados no nosso novo servidor. Para isso, execute o comando abaixo, que irá gerar uma nova base de dados chamada “aspnetcore”:

|  |
| --- |
| ~$ createdb aspnetcore |

Nesse momento, é interessante abrir uma segunda janela do terminal para manter nosso servidor do PostgreSQL rodando. Portanto, abra outra janela do terminal, digite o comando a seguir e então minimize-a:

|  |
| --- |
| ~$ postgres –D /usr/local/var/postgres |

Feito isso, baixe e instale a ferramenta de administração [PSequel](http://www.psequel.com/" \t "_blank). Ela não é obrigatória para o funcionamento da aplicação com o banco de dados, mas nos ajuda a visualizar a estrutura do banco (veja a **Figura 1**).

  
**Figura 1.** Ferramenta de administração do PostgreSQL PSequel

Os dados de conexão do nosso banco de dados são:

* **Host:** localhost (padrão);
* **Porta:**5432 (padrão);
* **Database:** aspnetcore (base que criamos anteriormente);

Nesse instante, o banco de dados estará vazio. Iremos desenvolver, portanto, a aplicação que se conectará a ele.

### Passo 2: Criação do projeto ASP.NET Core e adição de dependências

Agora que nosso ambiente de desenvolvimento já dispõe do PostgreSQL, vamos criar nossa aplicação e chamá-la de DataWebApp, utilizando para isso os seguintes comandos:

|  |
| --- |
| ~$ mkdir DataWebApp  ~$ cd DataWebApp  ~$ dotnet new –t web |

Com a aplicação criada, abra-a no seu editor para começarmos as mudanças. A primeira coisa que temos que fazer é instalar o pacote NuGet com a biblioteca necessária para usarmos o PostgreSQL na aplicação. Sendo assim, no arquivo project.json, presente na raiz do projeto, na seção de dependencies, adicione a seguinte linha:

|  |
| --- |
| "Npgsql.EntityFrameworkCore.PostgreSQL": "1.0.0" |

Logo após, salve as alterações, volte ao terminal e digite:

|  |
| --- |
| ~$ dotnet restore  ~$ bower install |

A primeira linha realizará o download de todos os pacotes NuGet dos quais nossa aplicação depende, os quais estão presentes no arquivo project.json. Já a segunda linha realiza o download das dependências de front-end da nossa aplicação – como o Bootstrap e o jQuery – utilizando um gerenciador de pacotes chamado Bower, cujas dependências são configuradas no arquivo bower.json.

***Observação: Caso sua máquina não possua o Bower instalado, digite o seguinte comando no Terminal:***

|  |
| --- |
| ~$ npm install –g bower |

E caso você também não possua o NPM, basta fazer o download do instalador mais recente no [site oficial](http://nodejs.org/).

http://www.devmedia.com.br/layout/icon/information-button.png Saiba mais sobre [como criar projetos ASP.NET Core no MacOS](http://www.devmedia.com.br/criando-projetos-asp-net-core-no-macos/36770)

### Passo 3: Configuração da biblioteca do PostgreSQL

Neste momento o banco de dados já está instalado e criado, e a aplicação já tem a capacidade de se conectar a ele. Então, podemos fazer a configuração dessa conexão. Para isso, primeiramente devemos abrir o arquivo appsettings.json, que possui alguns parâmetros de configuração, e modificar a string de conexão em Default Connection, para a seguinte:

|  |
| --- |
| “Default Connection”: “Host=localhost;Port=5432;Database=aspnetcore” |

***Observação: Essa string de conexão segue o padrão do PostgreSQL. Caso queira saber sobre mais parâmetros que podem ser incluídos, visite a***[***documentação oficial***](http://www.npgsql.org/doc/connection-string-parameters.html)***.***

Após isso, podemos abrir o arquivo Startup.cs, que é responsável por configurar todos os serviços e todos os módulos requeridos pela nossa aplicação, e substituir a dependência do SQLite pelo PostgreSQL. No método ConfigureServices(IServiceCollection services), localize a seguinte linha:

|  |
| --- |
| services.AddDbContext<ApplicationDbContext>(options =>         options.UseSqlite(Configuration.GetConnectionString("DefaultConnection"))); |

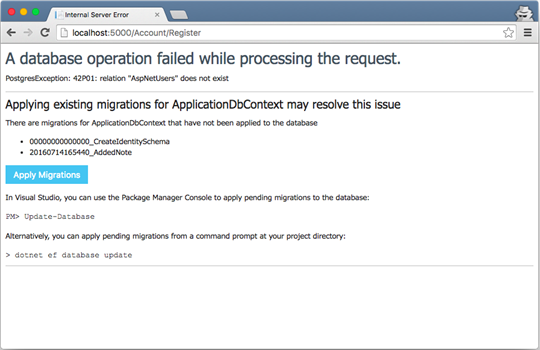
Em seguida, a substitua por:

|  |
| --- |
| services.AddDbContext<ApplicationDbContext>(options =>           options.UseNpgsql(Configuration.GetConnectionString("DefaultConnection"))); |

Perceba que a única mudança de código foi feita em UseNpgsql, que diz para a aplicação usar o conector do PostgreSQL ao invés do conector do SQLite. Esse método de extensão foi disponibilizado no momento em que instalamos o pacote Npgsql.EntityFrameworkCore.PostgreSQL pelo NuGet.

### Passo 4: Atualizando o banco de dados

Agora, ao executar a nossa aplicação utilizando o comando **dotnet run**, e a acessar pelo endereço http://localhost:5000, tudo aparentará funcionar normalmente. No entanto, se clicarmos em Registere tentarmos criar uma conta, nos depararemos com a tela de erro ilustrada na **Figura 2**.

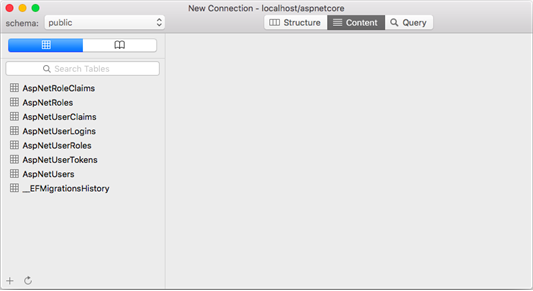
  
**Figura 2.** Erro na aplicação referente à execução da migrations

É importante lembrar que nossa aplicação já vem com o ASP.NET Identity instalado – o serviço de autenticação e autorização do ASP.NET – e que ele depende de algumas tabelas que devem ser criadas no banco de dados. Para facilitar esse processo, existe, entre nossa aplicação e o banco de dados, uma camada de abstração com o Entity Framework (EF). Esse possui um modelo de desenvolvimento chamado Code-First, que determina que as classes que mapeiam as entidades do banco de dados devem ser criadas primeiro, para depois o EF traduzi-las para código SQL e, posteriormente, tabelas e dados.

A estrutura de classes necessárias ao Identity, no entanto, já foi criada automaticamente quando iniciamos nosso projeto. Resta agora executarmos o comando para o EF criar as tabelas no banco. Dito isso, na janela do terminal, digite a seguinte instrução:

|  |
| --- |
| ~$ dotnet ef database update |

Neste momento, se voltarmos ao PSequel e olharmos para o banco (**Figura 3**), veremos que ele está preenchido com as tabelas. Podemos, então, fazer o registro de uma nova conta de usuário na nossa aplicação web normalmente, utilizando as funções que são criadas por padrão, uma vez que a conexão com o banco de dados já foi estabelecida.

  
**Figura 3.** Banco de dados preenchido com as tabelas do ASP.NET Identity

A partir disso, podemos criar as entidades da aplicação, realizar o mapeamento delas com o Entity Framework e, logo após, gerar a estrutura do banco de dados por meio das migrations.

http://www.devmedia.com.br/layout/icon/information-button.png Saiba mais sobre [Migrations no Entity Framework Core](http://www.devmedia.com.br/entity-framework-core-criando-bases-de-dados-com-migrations/36776" \t "_blank)