

FULL STACK DEVELOPMENT

FASE 2 | AULA 01 -

CONEXÃO COM BANCO DE DADOS POSTGRESQL

SUMÁRIO

O QUE VEM POR AÍ?	3
HANDS ON	4
SAIBA MAIS	5
O QUE VOCÊ VIU NESTA AULA?	10
REFERÊNCIAS.....	11

EXEMPLO

O QUE VEM POR AÍ?

Boas-vindas à nossa aula sobre conexão com banco de dados, onde exploraremos a integração com bancos de dados PostgreSQL, usando tanto o driver nativo quanto o Prisma ORM. Esta disciplina abrangerá desde a configuração inicial até a construção de uma API robusta e sua posterior testagem.

Ao longo desta jornada, mergulharemos no universo do NestJS, um framework Node.js que nos proporciona uma arquitetura modular, baseada em módulos e decoradores. Vamos explorar a conexão com o PostgreSQL, tanto através do driver nativo quanto do Prisma ORM, oferecendo uma visão abrangente de ambas as abordagens.

Mas não pararemos por aí! Para garantir um ambiente de desenvolvimento consistente e isolado, utilizaremos containers Docker para configurar nosso banco de dados PostgreSQL. Aprenderemos como criar, iniciar e gerenciar containers para facilitar o desenvolvimento e a distribuição de nossas aplicações.

Ao finalizar a configuração do ambiente, partiremos para a construção de uma API eficiente e escalável com NestJS. Exploraremos conceitos essenciais, como controladores, serviços, e módulos, aplicando boas práticas de desenvolvimento.

Por fim, vamos garantir a qualidade do nosso código ao realizar testes na nossa API através de uma interface gráfica que é o Insomnia.

Sejam bem-vindos(as) a esta emocionante jornada de aprendizado! Vamos explorar, construir e testar, te capacitando para criar aplicações Node.js utilizando NestJS e integrando com banco de dados.

HANDS ON

Embarcaremos em uma jornada prática para configurar nosso ambiente de desenvolvimento, construir uma aplicação NestJS e integrá-la com um banco de dados PostgreSQL. Vamos começar com a configuração do ambiente, garantindo que todos e todas estejam prontos(as) para mergulhar no mundo do NestJS.

Em seguida, instalaremos a CLI do NestJS, uma ferramenta poderosa que facilita a criação de projetos, módulos e componentes. Criaremos nosso projeto inicial utilizando a CLI, estabelecendo assim as bases para a construção de uma aplicação robusta.

Daremos um mergulho profundo na conexão com o banco de dados PostgreSQL, começando com o uso do driver nativo para Node.js. Veremos como configurar essa conexão, criar consultas e interagir com nosso banco de dados de forma direta.

Em seguida, daremos um salto em direção à implementação da camada de controller, onde organizaremos nossas rotas e a lógica de manipulação de dados. Vamos construir as chamadas ao banco de dados de maneira estruturada e eficiente, preparando o terreno para uma API poderosa.

E como ponto culminante, faremos a transição para a utilização do Prisma ORM. Você aprenderá como integrar o Prisma ao NestJS, simplificando a interação com o banco de dados e beneficiando-se das funcionalidades avançadas que o ORM oferece.

SAIBA MAIS

Em meio ao intrigante desenvolvimento da tecnologia, a ligação entre aplicações e bancos de dados surge como uma necessidade importante que se manteve ao longo do tempo. Ao estudarmos os primórdios da computação nas décadas de 1950 e 1960, as aplicações da época eram, em sua maioria, monolíticas, operando em sistemas centralizados.

À medida que a quantidade de dados a ser manipulada crescia exponencialmente, tornou-se evidente que uma abordagem mais estruturada era crucial para armazenar e recuperar informações eficientemente. Assim nasceu a demanda por conectar aplicações a Sistemas de Gerenciamento de Banco de Dados (SGBD). Esses sistemas ofereciam uma estrutura organizada para armazenar dados, facilitando consultas complexas e proporcionando uma manipulação mais eficiente.

Ao longo das décadas, com o avanço tecnológico e a explosão de dados na era digital, a necessidade de comunicação eficaz entre aplicações e bancos de dados persistiu. A evolução da internet intensificou essa necessidade, e o cenário atual reflete uma integração massiva, impulsionada por diversos fatores.

Vivemos na era do big data, onde a geração colossal de informações exige uma interação constante entre aplicações e bancos de dados. A diversidade de aplicações, desde sistemas empresariais complexos até aplicativos móveis simples, amplifica a demanda por uma comunicação eficiente com bancos de dados. Empresas modernas, dependentes de dados para decisões estratégicas, encontram na integração contínua a agilidade necessária para se manterem competitivas.

Nesse cenário, a integração entre aplicações e bancos de dados não é apenas uma característica; é um alicerce essencial para a eficiência operacional e a tomada de decisões informadas. Ao compreender a trajetória histórica dessa conexão, fica evidente que ela não apenas sobreviveu, mas floresceu, adaptando-se dinamicamente às demandas crescentes e às oportunidades que o universo da tecnologia oferece.

O PostgreSQL, também conhecido como Postgres, é um sistema de gerenciamento de banco de dados relacional (RDBMS) de código aberto. Sua

arquitetura robusta e extensível o torna uma escolha popular para uma ampla gama de aplicativos, desde pequenos projetos até grandes implementações empresariais.

Temos algumas features que encantam no PostgreSQL, por isso a adoção dele vem se tornando cada vez mais comum. Entre elas, podemos citar:

- **Extensibilidade:** o PostgreSQL permite a criação de tipos de dados, funções e até mesmo linguagens de programação adicionais, proporcionando uma flexibilidade inigualável.
- **Suporte a Tipos Complexos:** além dos tipos de dados comuns, o Postgres suporta tipos complexos, como arrays, JSON, hstore, e até mesmo tipos personalizados.
- **Integridade Referencial e Transações ACID:** oferece suporte a transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade) e mantém a integridade referencial para garantir a confiabilidade dos dados.
- **Desempenho e Otimização:** possui otimizador de consultas avançado, índices eficientes e ferramentas para ajuste fino, proporcionando desempenho excepcional.
- **Replicação e Alta Disponibilidade:** suporta diferentes formas de replicação, permitindo a criação de ambientes de alta disponibilidade e recuperação de desastres.

O PostgreSQL traz diversas vantagens estratégicas. Como sua licença BSD permite seu uso gratuito, incentivando a comunidade e a colaboração, isso fomenta a comunidade global a oferecer suporte, recursos e atualizações frequentes, mantendo o PostgreSQL relevante e atualizado. Ele também traz compatibilidade SQL e Padrões, aderindo rigorosamente aos padrões SQL, facilitando a migração de outros sistemas de banco de dados relacionais.

No universo dinâmico dos bancos de dados, o PostgreSQL se destaca como uma escolha especialmente atrativa para empresas de todos os portes. Sua característica de ser open source, combinada com uma gama abrangente de recursos avançados, cria uma equação de custo-benefício irresistível. Esse fator tem

impulsionado a adoção em larga escala, permitindo que empresas alcancem suas metas de dados de forma econômica.

Outro pilar fundamental do PostgreSQL é a sua arquitetura escalável, um trunfo valioso para empresas em crescimento. A confiança de que o desempenho não será comprometido à medida que a organização se expande é um diferencial significativo. Essa adaptabilidade dinâmica é um fator crítico, fornecendo às empresas a flexibilidade necessária para evoluir em um ambiente de negócios em constante mudança.

Seja para impulsionar aplicativos web, conduzir análises de dados, lidar com geoespacialidade ou integrar-se à Internet das Coisas (IoT), o PostgreSQL se posiciona como um parceiro confiável, moldando-se aos requisitos específicos de cada cenário.

Se você está ansioso(a) para aprofundar seus conhecimentos, uma leitura altamente recomendada é "Mastering PostgreSQL in Application Development" de Dimitri Fontaine. Este livro não apenas explora os aspectos técnicos, mas também oferece insights práticos sobre como aproveitar ao máximo o PostgreSQL em desenvolvimento de aplicativos.

Para uma fonte oficial e atualizada, visite o site oficial do PostgreSQL. Lá, você encontrará documentação detalhada, notícias e recursos para continuar sua jornada de aprendizado. Se a abordagem interativa é mais a sua cara, o PostgreSQL Tutorial é uma excelente escolha para práticas e experimentos guiados.

E falando de PostgreSQL, não podemos nos esquecer do Docker que utilizamos nessa aula. O Docker emergiu como uma resposta engenhosa para um desafio persistente: a consistência e a portabilidade de ambientes de desenvolvimento e produção. Antes do Docker, pessoas desenvolvedoras frequentemente enfrentavam o dilema de garantir que suas aplicações funcionassem da mesma forma em diferentes máquinas e ambientes. Cada sistema operacional, configuração e dependência adicionada à complexidade.

O Docker foi concebido para solucionar essa questão, fornecendo uma solução de virtualização de contêineres. Ele encapsula aplicações e suas dependências em contêineres isolados, garantindo que elas possam ser executadas de maneira consistente em qualquer ambiente que suporte o Docker. Essa abordagem resolveu

o problema da famosa frase "Funciona na minha máquina", tornando as aplicações altamente portáteis.

A história do Docker começou em 2013, quando Solomon Hykes lançou a ferramenta. Desde então, o Docker se tornou uma tecnologia fundamental no mundo do desenvolvimento de software. Seus contêineres são leves, rápidos e compartilham recursos do sistema operacional subjacente, tornando-os eficientes e escaláveis.

Para pessoas desenvolvedoras, a utilidade do Docker reside em sua simplicidade. Os principais comandos são acessíveis, mesmo para aqueles que não têm uma profunda compreensão de virtualização. Para iniciar, o comando `docker run` é a porta de entrada para criar e executar contêineres. Em seguida, `docker build` permite a criação de imagens personalizadas. Para gerenciar esses contêineres em execução, `docker ps` exibe uma lista, e `docker stop` ou `docker rm` encerra ou remove contêineres, respectivamente.

Quando executamos `docker run`, podemos adicionar algumas opções úteis. A flag `-d` é uma abreviação de "detach", permitindo que o contêiner seja executado em segundo plano, liberando a linha de comando para outras operações. Já a flag `-t` indica que queremos alocar um pseudo-TTY, tornando a interação com o contêiner mais amigável.

Quando tratamos de segurança e configuração, é comum precisarmos passar variáveis de ambiente, como senhas, durante a inicialização do contêiner. Isso é feito com a opção `-e`. Por exemplo, ao lidar com um banco de dados, podemos definir a variável de ambiente `POSTGRES_PASSWORD` para especificar a senha. A linha de comando ficaria algo como:

```
"docker run -d -t -e POSTGRES_PASSWORD=sua_senha postgres:latest"
```

Neste exemplo, estamos iniciando um contêiner PostgreSQL em segundo plano, alocando um pseudo-TTY e definindo a senha do banco de dados usando a variável de ambiente `POSTGRES_PASSWORD`. Isso proporciona uma maneira prática e segura de passar informações sensíveis para o contêiner durante a inicialização.

Outro cenário onde podemos utilizar o Docker é com nossa aplicação NodeJS, para executá-la em um ambiente isolado, para desenvolvimento ou produção. O que acha do desafio de tentar "dockerizar" a aplicação que criamos com NestJS?

E falando de NestJS, ele é um framework para construção de aplicativos back-end em Node.js, que se destaca por sua arquitetura modular e orientada a serviços. Criado por Kamil Myśliwiec e mantido pela comunidade, o NestJS foi desenvolvido com a missão de fornecer uma estrutura robusta para facilitar o desenvolvimento de aplicações escaláveis e eficientes.

Por baixo dos panos, o NestJS utiliza o TypeScript, uma extensão do JavaScript, que adiciona tipos estáticos à linguagem. Essa escolha traz benefícios significativos, como maior segurança e produtividade durante o desenvolvimento. O uso do Typescript traz a vantagem de permitir a definição de tipos e interfaces, proporcionando um desenvolvimento mais seguro e autodocumentado. Além disso, ele adota o padrão de projeto Dependency Injection, facilitando a gestão de dependências e tornando o código mais modular e fácil de entender.

Além disso, o NestJS faz uso do Express.js, um framework web para Node.js, para lidar com as camadas de requisição e resposta HTTP.

O NestJS acelera o desenvolvimento ao adotar uma arquitetura modular e bem estruturada, inspirada no Angular. Utiliza o conceito de módulos, controladores e serviços para organizar o código de maneira coesa e reutilizável. Os controladores definem rotas e recebem requisições HTTP, enquanto os serviços encapsulam a lógica de negócios e podem ser injetados em diversos componentes.

A arquitetura do NestJS é fortemente baseada no padrão de projeto MVC (Model-View-Controller), mas com uma abordagem mais orientada a serviços. A separação clara de responsabilidades facilita a manutenção do código, testes unitários e a escalabilidade da aplicação.

Ao unir todos esses conceitos e práticas, o NestJS oferece não apenas uma estrutura para o desenvolvimento, mas também promove uma mentalidade e um conjunto de padrões que elevam a qualidade e a organização do código. Essa abordagem coesa e modular não apenas acelera o desenvolvimento, mas também contribui para a manutenção a longo prazo, tornando o NestJS uma escolha popular para profissionais de desenvolvimento que buscam eficiência e escalabilidade em suas aplicações Node.js.

Conhecendo a parte prática do NestJS e agora a parte teórica, o que acha de finalizar as rotas de update e delete que ficaram faltando no nosso projeto?

O QUE VOCÊ VIU NESTA AULA?

Nessa aula aprendemos sobre a integração com PostgreSQL, foi uma imersão valiosa em todo o ciclo de desenvolvimento de uma aplicação back-end, desde a criação e configuração de containers Docker para o PostgreSQL até a exploração de ORMs e a execução de testes manuais com o Insomnia, cada passo contribuiu para entendermos mais como tudo se conecta.

A experiência prática de criar um projeto com NestJS e as facilidades que esse framework trouxe nos mostraram como é simples e rápido criar uma API.



REFERÊNCIAS

NIELD, Thomas. **Introdução à Linguagem SQL: Abordagem prática para iniciantes**. São Paulo: novatec, 2016.

GRONER, Loiane. **Estruturas de Dados e Algoritmos com JavaScript: Escreva um Código JavaScript Complexo e Eficaz Usando a Mais Recente ECMAScript: Abordagem prática para iniciantes**. São Paulo: novatec, 2019.

CARVALHO, Vinícius. **PostgreSQL: Banco de dados para aplicações web modernas: Abordagem prática para iniciantes**. São Paulo: Casa do Código, 2017.

PALAVRAS-CHAVE

Palavras-chave: NodeJS. PostgresSQL. NestJS.API. SQL. BACK-END.

EMSE

POS TECH