Iniciando com o Node.js

Iniciamos nossa jornada no universo do Node.js, uma etapa fundamental para quem deseja aprofundar-se em programação. Neste módulo, nosso foco será uma introdução detalhada ao NPM (Node Package Manager), o gerenciador de pacotes que é peça-chave no desenvolvimento com Node. A compreensão de como o NPM funciona e sua capacidade de gerenciar as dependências de um projeto não apenas facilitará a configuração e manutenção de suas aplicações, mas também otimizará seu fluxo de trabalho de maneira significativa.

NPM

O NPM destaca-se como a ferramenta essencial para qualquer desenvolvedor que trabalhe com Node.js. Ele não só permite iniciar projetos, administrar pacotes e resolver questões de dependências desatualizadas de forma automatizada, mas também economiza um tempo valioso que, de outra forma, seria gasto em tarefas manuais e repetitivas. Com o NPM, você tem à disposição um ecossistema rico de pacotes que podem ser facilmente adicionados ao seu projeto, proporcionando uma vasta gama de funcionalidades prontas para uso.

Neste módulo, além de introduzir o conceito e a importância do NPM, abordaremos como utilizá-lo na prática, desde a configuração inicial de um projeto Node.js até a gestão eficaz de suas dependências. Serão explorados comandos fundamentais como `npm init`, que inicia um novo projeto Node.js, e `npm install`, utilizado para adicionar novos pacotes ao seu projeto. Esses comandos são a base para construir aplicações robustas e manter seu projeto atualizado e seguro.

A utilização do NPM é crucial não apenas para desenvolvedores iniciantes, mas também para aqueles que já atuam profissionalmente. Entender como o NPM funciona e como aplicá-lo em seus projetos é uma habilidade valiosa, que certamente trará benefícios tanto para seus projetos acadêmicos quanto profissionais. Com ele, você pode gerenciar eficientemente as dependências de sua aplicação, assegurando que todos os pacotes utilizados estejam na versão correta e atualizados, evitando assim possíveis incompatibilidades e vulnerabilidades.

À medida que avançamos neste módulo, você conseguirá aplicar os conceitos aprendidos em situações práticas, preparando-se para os próximos tópicos que incluem o uso de export e import, a diferenciação entre módulos internos e externos, a compreensão dos módulos para interação e requisições, e, finalmente, a apresentação de um mini projeto. Esse projeto servirá como base para aplicar o conhecimento adquirido e explorar como essas técnicas e ferramentas podem ser reaproveitadas em seu portfólio ou no ambiente profissional.
Concluímos, assim, a introdução ao NPM, um componente vital no desenvolvimento com Node.js. Com essa base estabelecida, estamos prontos para mergulhar ainda mais profundamente nos aspectos técnicos e práticos do Node.js. Mantenha seu entusiasmo e prepare-se para explorar a riqueza de possibilidades que o desenvolvimento com Node.js oferece.

Uso de export e import

Dando continuidade, vamos nos aprofundar nos conceitos de exportação e importação de módulos, fundamentais para o desenvolvimento de aplicações eficientes e organizadas. Apesar de parecerem conceitos simples à primeira vista, entender profundamente o uso de `export` e `import` é essencial antes de avançarmos nas práticas de Node.js.

No Node.js, cada arquivo de código pode ser visto como um módulo separado, permitindo uma modularização do código que facilita a

manutenção, a leitura e a reutilização de funcionalidades. Para que um módulo possa disponibilizar suas funções, variáveis ou objetos para outros módulos dentro da aplicação, ele deve exportá-los usando `module.exports`. Este passo é crucial, pois define explicitamente quais partes do módulo podem ser acessadas externamente.

Por exemplo, suponha que você tenha criado um módulo que realiza operações matemáticas básicas. Para tornar essas funcionalidades acessíveis a outros arquivos do seu projeto, você utilizará algo semelhante a:

```
// operacoes.js
function somar(a, b) {
return a + b;
}

function subtrair(a, b) {
return a - b;
}

module.exports = { somar, subtrair };
```

Em contrapartida, quando você deseja utilizar as funcionalidades disponibilizadas por um módulo, você as importa em outro arquivo através do comando `require`, indicando o caminho do módulo que deseja acessar. Continuando com nosso exemplo, para usar as funções de `operacoes.js` em outro arquivo, faríamos:

```
// app.js
const operacoes = require('./operacoes');

console.log(operacoes.somar(5, 3));
console.log(operacoes.subtrair(10, 5));
```

Essa interação entre `export` e `import` não se limita apenas aos módulos criados por você, mas também se aplica ao vasto ecossistema de pacotes disponíveis através do NPM, reforçando a importância de compreender como esses comandos funcionam. Ao importar pacotes, você está, na essência, incorporando módulos externos exportados por outros desenvolvedores em sua aplicação, ampliando significativamente as funcionalidades do seu projeto sem a necessidade de reescrever código já existente.

Portanto, dominar o fluxo de exportar suas próprias funcionalidades e importar as necessárias, seja de módulos internos, seja de pacotes externos, é uma habilidade valiosa no desenvolvimento Node.js. Esse conhecimento será aplicado em práticas subsequentes, onde você terá a oportunidade de estruturar seu projeto de maneira eficaz, utilizando importações para compor a lógica da sua aplicação e exportações para modularizar seu código.

Módulos Internos e Externos

Prosseguindo, adentramos agora no estudo dos módulos internos e externos, elementos cruciais para a construção de aplicações robustas e escaláveis. Ao entender a mecânica por trás da exportação e importação de módulos, estamos prontos para aprofundar nosso conhecimento nessa área fundamental.

Os módulos em Node.js são essencialmente blocos de construção de código que permitem a segmentação lógica e funcional das aplicações. Cada arquivo JavaScript em um projeto Node.js é tratado como um módulo, que pode ser exportado e, consequentemente, importado por outros módulos. Isso não apenas promove a reutilização de código, mas também a manutenção e organização do projeto.

Existem três tipos principais de módulos no ecossistema do Node.js: módulos nativos, públicos e criados pelo usuário.

1. Módulos Nativos: são aqueles fornecidos pelo próprio Node.js, como `http`, `fs`, `path`, entre outros. Esses módulos vêm embutidos na instalação do Node.js, portanto não requerem instalação adicional através do NPM. A simplicidade de uso, mediante a importação direta, os torna uma escolha imediata para muitas funcionalidades comuns de baixo nível.

2. Módulos Públicos: disponibilizados através do registro do NPM, esses módulos cobrem uma vasta gama de funcionalidades, desde frameworks de desenvolvimento web como Express até bibliotecas para manipulação de dados como o Mongoose para MongoDB. Esses módulos são desenvolvidos, mantidos e disponibilizados pela comunidade, permitindo que desenvolvedores construam sobre o trabalho de outros e contribuam para o ecossistema.

3. Módulos Criados pelo Usuário: são módulos específicos de cada projeto, escritos pelos próprios desenvolvedores. Estes podem ser utilizados para organizar o código da aplicação, dividindo-o em partes menores e reutilizáveis, como serviços, utilitários e modelos. Isso facilita a organização lógica da aplicação e a reutilização de código dentro do mesmo projeto ou entre projetos diferentes.
Para ilustrar a prática, criaremos um servidor básico utilizando o módulo `http` nativo do Node.js. Primeiramente, inicializamos um projeto Node.js com o comando `npm init`, e após a configuração básica, procedemos à criação de um arquivo, por exemplo, `server.js`. Neste arquivo, importamos o módulo `http` e utilizamos sua função `createServer` para definir e iniciar um servidor HTTP simples:

```
const http = require('http');

const server = http.createServer((req, res) => {
```

```
res.statusCode = 200;
res.setHeader('Content-Type', 'text/plain');
res.end('Olá, Mundo!');
});

const PORT = 3000;
server.listen(PORT, () => {
console.log(`Servidor rodando em http://localhost:${PORT}/`);
});
```

Este exemplo destaca a facilidade com que os módulos nativos podem ser utilizados para criar funcionalidades complexas com poucas linhas de código. Ao executar este arquivo com `node server.js`, um servidor HTTP é iniciado, acessível através do navegador em `http://localhost:3000`, exibindo a mensagem "Olá, Mundo!".

Em síntese, os módulos internos e externos no Node.js desempenham um papel vital na estruturação e desenvolvimento de aplicações. Compreender como esses módulos funcionam, como são utilizados e como você pode criar seus próprios módulos, é fundamental para qualquer desenvolvedor trabalhando com Node.js.

Módulos Para Interação

Avançando em nossa exploração do Node.j, neste tópico, mergulharemos nos módulos para interação, uma parte crucial para o desenvolvimento de aplicações que manipulam arquivos e diretórios. Esta seção é fundamental para entender como o Node.js permite trabalhar com arquivos de diversos formatos de maneira eficiente e adaptável às necessidades de sua aplicação.

Os módulos para interação, como o próprio nome sugere, são ferramentas que o Node.js disponibiliza para facilitar a interação com o sistema de

arquivos e diretórios do seu computador. Entre os principais módulos, destacam-se o `fs` para o sistema de arquivos, o *path* para manipulação de caminhos de arquivos e diretórios, e o *inquirer* para criar interfaces de interação no terminal.

1. Módulo *fs* (File System): é o módulo nativo do Node.js usado para trabalhar com arquivos no sistema. Ele oferece uma ampla gama de funcionalidades, como leitura, escrita, modificação e exclusão de arquivos. Este módulo é fundamental para qualquer operação que envolva manipulação de arquivos, sendo uma das ferramentas mais utilizadas para tal fim.

2. Módulo *path*: facilita a manipulação de caminhos de arquivos e diretórios, ajudando a construir caminhos que sejam compatíveis com diferentes sistemas operacionais. Ele fornece métodos para extrair informações de caminhos de arquivo, como o diretório base, extensão do arquivo, e o nome do arquivo.

3. Módulo *inquirer*: utilizado para criar interfaces de linha de comando interativas. Ele permite que os desenvolvedores façam perguntas ao usuário, coletando inputs que podem ser usados para guiar a lógica de execução da aplicação.

Para exemplificar o uso prático desses módulos, focaremos no módulo *path* e realizar um exercício simples no Visual Studio Code. Suponha que desejamos identificar o nome base de um arquivo específico em um diretório. Para isso, iniciaremos criando um novo arquivo, denominado `path.js`, em uma pasta já configurada com o Node.js e gerenciada pelo NPM.

const path = require('path');

let nomeArquivo = path.basename('/caminho/para/seu/arquivo/teste.txt');

console.log(nomeArquivo);

Neste exemplo, utilizamos a função *basename* do módulo *path* para extrair o nome base do arquivo especificado. Isso demonstra como, com apenas algumas linhas de código, é possível realizar operações complexas de manipulação de arquivos e diretórios.

A prática ilustrada acima evidencia a simplicidade e a potência dos módulos para interação disponíveis no Node.js. Ao compreender e utilizar esses módulos, você será capaz de realizar tarefas complexas de maneira eficiente, desde a manipulação de arquivos e diretórios até a criação de interfaces de usuário interativas no terminal.

Concluímos, assim, nossa abordagem sobre módulos para interação no Node.js. A seguir, avançaremos para explorar módulos focados em requisições, ampliando nosso entendimento sobre o desenvolvimento de aplicações web com Node.js.

Módulos Para Requisições

Dando continuidade, exploraremos um aspecto vital para o desenvolvimento de aplicações web: os módulos para requisições. Este conhecimento é crucial quando nos debruçamos sobre aplicações que necessitam interagir com outras aplicações ou serviços web, executando requisições e processando respostas.

Os módulos para requisições são aqueles que nos ajudam a criar e gerenciar tanto requisições enviadas a servidores externos quanto as respostas recebidas. No ecossistema Node.js, dois dos módulos mais relevantes para esse fim são o *http* e o *url*.

1. **Módulo *http*:** já introduzido em práticas anteriores, este módulo é essencial para criar servidores HTTP locais. Ele nos permite escutar

requisições em portas específicas e responder a elas, essencial para qualquer aplicação web desenvolvida com Node.js.

2. Módulo *`url:`* este módulo fornece utilitários para a análise e manipulação de URLs. É particularmente útil para extrair partes específicas de uma URL ou construir URLs dinamicamente, baseadas em diferentes condições ou entradas.

Para ilustrar a aplicação desses módulos, realizaremos uma prática simples. Vamos configurar uma aplicação Node.js que responde com páginas HTML diferentes com base na URL acessada. Isso demonstrará como o Node.js pode ser utilizado para criar aplicações web dinâmicas.

Para começar, criaremos dois arquivos HTML simples, *`Inverno.html`* e *`Verão.html`*, representando diferentes conteúdos que nossa aplicação pode servir. Em seguida, desenvolveremos um script `escolha.js` que utilizará os módulos *`http`*, *`url`* e *`fs`* (para leitura de arquivos) para servir o conteúdo HTML apropriado com base na URL acessada.

No nosso script, utilizaremos o método *`createServer`* do módulo *`http`* para criar um servidor. O módulo *`url`* será usado para analisar a URL da requisição e decidir qual arquivo HTML servir. Utilizaremos a função *`readFile`* do módulo *`fs`* para ler o conteúdo do arquivo HTML correspondente e enviá-lo como resposta à requisição.

Esse exemplo simples nos permite observar o poder do Node.js para o desenvolvimento de aplicações web, demonstrando a facilidade com que podemos construir um servidor que serve conteúdo dinâmico baseado em URLs.

Apresentação do Mini Projeto

Chegamos à sexta e última parte de nossa aula introdutória ao Node.js na disciplina, onde daremos um grande passo em direção à aplicação prática

dos conceitos que exploramos até agora. Neste segmento, teremos o mini projeto que será o foco do nosso estudo e desenvolvimento ao longo desta disciplina.

Este projeto visa consolidar os conhecimentos adquiridos em um contexto prático, oferecendo a você a oportunidade de aplicar as ferramentas, frameworks e linguagens que introduzimos nas aulas anteriores. Para isso, usaremos o Node.js para estruturar o back-end de nossa aplicação e o framework Next.js para desenvolver o front-end, proporcionando uma experiência completa de desenvolvimento full-stack.

O Projeto: o projeto proposto é um sistema de gerenciamento para consultórios médicos, uma aplicação web que permite administrar informações relacionadas a médicos, pacientes, consultas e prescrições médicas. Este sistema será desenvolvido seguindo boas práticas de mercado, com código em inglês, permitindo que você se familiarize com o ambiente de desenvolvimento profissional.

O sistema contará com quatro entidades principais:

- **Doutores (Doctors):** profissionais de saúde que poderão acessar o sistema para gerenciar suas consultas e prescrições;
- **Pacientes (Patients):** indivíduos que recebem atendimento médico e têm suas informações registradas no sistema;
- **Consultas (Appointments):** registros das visitas dos pacientes aos médicos, incluindo data, horário e detalhes da consulta;
- **Prescrições (Prescriptions):** detalhes das medicações e instruções prescritas pelos médicos aos pacientes.

Estas entidades estarão inter-relacionadas de maneira que um doutor possa ter múltiplas consultas agendadas, cada consulta pode gerar várias prescrições, e um paciente pode ter um histórico de múltiplas consultas. O design e a lógica do sistema deverão refletir e facilitar o gerenciamento eficiente dessas relações.

Para o desenvolvimento, focaremos inicialmente no back-end, construindo a lógica de negócios e a API com Node.js. Gradualmente, integraremos o Next.js para criar uma interface de usuário amigável e intuitiva. Ao longo do processo, aplicaremos práticas de desenvolvimento como o uso de NPM para gestão de pacotes, implementação de testes unitários para garantir a qualidade do código, e o uso de bancos de dados não relacionais para armazenamento de dados.

Além de ser uma ótima adição ao seu portfólio, este projeto oferecerá insights valiosos sobre como os conceitos e ferramentas estudados podem ser aplicados em cenários reais de desenvolvimento. A estrutura do projeto e todas as informações necessárias estarão disponíveis no GitHub da disciplina, garantindo que você tenha acesso a todos os recursos necessários para o desenvolvimento.

Ao final deste projeto, você terá uma compreensão sólida de como construir uma aplicação web completa usando Node.js e Next.js, além de uma apreciação prática das nuances envolvidas no desenvolvimento de sistemas de informação complexos.

Na próxima aula, começaremos a mergulhar na implementação desse projeto, passo a passo. Até lá, encorajo você a revisar os conceitos que cobrimos até agora e a se preparar para a empolgante jornada de desenvolvimento que temos pela frente. Até mais, e que seu projeto seja um sucesso!

GitHub da Disciplina

Você pode acessar o repositório da disciplina no GitHub a partir do seguinte link: https://github.com/FaculdadeDescomplica/ProgramacaoII. Esse repositório tem como principal objetivo guardar os códigos das aulas práticas da disciplina para aprimorar suas habilidades em vários tópicos, incluindo a criação e consumo de APIs com controle de autenticação utilizando Node.js e utilizando boas práticas de programação e mercado.

Conteúdo Bônus

Após uma imersão profunda nos fundamentos do Node.js, desde a introdução ao NPM até a apresentação de nosso mini projeto, é momento de expandir ainda mais seu conhecimento. Para isso, sugiro como conteúdo bônus uma exploração detalhada de uma ferramenta amplamente utilizada no ecossistema Node.js: Express.js.

Express.js é um framework para Node.js que simplifica a criação de aplicações web e APIs. Sua flexibilidade e leveza fazem dele uma escolha popular entre desenvolvedores que buscam construir aplicações robustas e eficientes.

O que explorar sobre Express.js:

- Conceitos Básicos e Instalação: entenda o que faz do Express.js um framework tão poderoso para o desenvolvimento web em Node.js. Explore como instalá-lo em seu projeto e as principais razões para escolhê-lo.

- Rotas e Middleware: aprofunde-se na criação de rotas para manejar diferentes requisições HTTP e descubra o poder dos middlewares para expandir as funcionalidades das suas aplicações Express.

- Construção de APIs: veja como Express facilita a construção de APIs RESTful, permitindo que aplicações cliente, como aquelas construídas com React ou Angular, consumam dados de maneira eficaz.

- Segurança e Performance: explore práticas recomendadas para tornar suas aplicações Express mais seguras e performáticas.

Diversos autores e contribuidores têm escrito sobre o Express.js, mas para este conteúdo bônus, recomendamos focar em materiais produzidos por desenvolvedores experientes na comunidade Node.js, que compartilham dicas, tutoriais e melhores práticas.

Onde Pesquisar:

MDN Web Docs (Mozilla Developer Network): um recurso confiável que frequentemente oferece guias e tutoriais sobre diversas tecnologias web, incluindo Node.js e Express.js.

Documentação Oficial do Express.js: a documentação oficial é um excelente ponto de partida para qualquer desenvolvedor que deseje aprofundar-se no uso do Express. Ela oferece uma visão abrangente de todas as funcionalidades do framework, incluindo exemplos de código e explicações detalhadas.

Referências Bibliográficas

Bibliografia Básica:

ELMASRI, R.; NAVATHE, S. B. **Sistemas de banco de dados.** 7.ed. Pearson: 2018.

MEDEIROS, L. F. de. **Banco de dados:** princípios e prática. Intersaberes: 2013.

VICCI, C. (Org.). **Banco de dados**. Pearson: 2014.

Bibliografia Complementar:

CARDOSO, L. da C. **Design de aplicativos**. Intersaberes: 2022.

JOÃO, B. do N. **Usabilidade e interface homem-máquina**. Pearson: 2017.

LEAL, G. C. L. **Linguagem, programação e banco de dados**: guia prático de aprendizagem. Intersaberes: 2015.

PUGA, S.; FRANÇA, E.; GOYA, M. **Banco de dados**: implementação em SQL, PL/SQL e Oracle 11g. Pearson: 2013.

SETZER, V. W.; SILVA, F. S. C. **Bancos de dados**. Blucher: 2005.

Ir para exercício