# Trabalhando com arquivos

## Introdução

Na era digital atual, o manuseio eficiente de arquivos é uma habilidade indispensável no desenvolvimento de aplicações robustas e dinâmicas. Neste contexto, a disciplina de Programação II se propõe a aprofundar-se no tema "Trabalhando com arquivos", explorando conceitos fundamentais e práticas essenciais utilizando Node.js. Desde a gravação e leitura de arquivos até o upload de anexos, cada etapa desempenha um papel crucial na construção de sistemas informatizados que lidam com uma vasta quantidade de dados. Através deste módulo, navegaremos pelas ferramentas que o Node.js oferece, como o módulo FS para interação com o sistema de arquivos, destacando a importância de selecionar o módulo adequado para cada tarefa específica. Com uma abordagem teórica inicial, seguida de exemplos práticos, este segmento busca não apenas transmitir o conhecimento técnico necessário, mas também estimular a curiosidade e o pensamento crítico, preparando você para aplicar estas competências em projetos futuros. Juntos, exploraremos as possibilidades que a programação nos oferece, marcando o início de nossa jornada rumo ao domínio do trabalho com arquivos.

## Módulos NodeJS para trabalhar arquivos

Neste tópico, nos dedicaremos ao estudo e prática do trabalho com arquivos usando Node.js, um tópico essencial para o desenvolvimento de aplicações robustas e eficientes. O manuseio de arquivos é uma tarefa comum e crucial na programação, e exploraremos quatro práticas principais relacionadas a esse trabalho.

Primeiramente, vamos aprofundar nos módulos principais disponíveis no Node.js para o manuseio de arquivos. O módulo `fs` (File System) é o mais usado e será o foco da nossa atenção. Ele fornece funcionalidades que permitem criar, ler, escrever e deletar arquivos.

Como exemplo, ao trabalhar com a gravação de arquivos, podemos desenvolver um sistema de gerenciamento de notas de alunos. Utilizando o método `writeFile` do módulo `fs`, o código para criar um arquivo `notas.txt` e escrever as notas dos alunos seria:

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

const notas = 'Alice: 9.0, Bob: 7.5, Clara: 8.0';

fs.writeFile('notas.txt', notas, err => {

  if (err) throw err;

  console.log('Notas salvas com sucesso!');

});
```

Este exemplo demonstra a gravação de dados em um arquivo de texto, uma operação fundamental em diversas aplicações.

Avançando para o upload de arquivos, uma prática essencial em aplicações que interagem com documentos, imagens e outros tipos de arquivos, vamos explorar o `multer`, um middleware para o Express que facilita o manuseio de uploads de arquivos. Para implementar o upload de imagens de perfil em um sistema de gerenciamento de usuários, usamos o seguinte código:

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

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

```
const app = express();

const storage = multer.diskStorage({

  destination: (req, file, cb) => {

    cb(null, 'uploads/')

  },

  filename: (req, file, cb) => {

    cb(null, file.fieldname + '-' + Date.now())

  }

});

const upload = multer({ storage: storage });

app.post('/profile', upload.single('avatar'), (req, res)
=> {

  res.send('Imagem de perfil atualizada com sucesso!');

});

app.listen(3000, () => console.log('Servidor rodando na
porta 3000'));
```

Este código cria uma rota POST no Express para receber uploads de arquivos, usando o `multer` para definir o armazenamento e manipular os arquivos enviados.

Por último, a leitura de arquivos é outra operação essencial, permitindo acessar e recuperar dados de arquivos existentes. Utilizando o arquivo

notas.txt que criamos, o código para ler e exibir seu conteúdo é:

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

fs.readFile('notas.txt', 'utf8', (err, data) => {

  if (err) throw err;

  console.log('Conteúdo do arquivo:', data);

});
```

Este trecho de código demonstra a leitura de um arquivo, permitindo visualizar os dados gravados anteriormente.

Além do fs, exploraremos outros módulos e ferramentas que complementam as funcionalidades de trabalho com arquivos, adequando-se às necessidades específicas de cada projeto. Dependendo da tarefa, outros módulos como path para manipulação de caminhos de arquivos, ou stream para processamento de grandes volumes de dados em fluxo, podem ser necessários para uma implementação eficiente.

Continuaremos com a prática de gravação, leitura e upload de arquivos, utilizando o módulo fs e outras ferramentas auxiliares para enriquecer a funcionalidade de nossos projetos de programação.

**Gravação de arquivos**

Nesta parte da nossa aula, focaremos em como integrar a funcionalidade de gravação de arquivos em sua aplicação, utilizando o Node.js. O módulo nativo FS é a ferramenta chave para essa tarefa, oferecendo um método amplamente utilizado para escrever arquivos. Esse método, conhecido como WriteFile, é essencial por sua flexibilidade e simplicidade, permitindo a gravação de dados em arquivos com um manejo eficaz de erros.

Vamos explorar um exemplo prático que demonstra o uso do `WriteFile` para criar um novo arquivo. Esse método requer alguns parâmetros fundamentais, como o caminho do diretório, o nome do arquivo, o conteúdo a ser gravado, e uma função de callback para tratamento de erros. Essa abordagem é similar ao que já praticamos em outras partes da disciplina, como o mapeamento de rotas e a interação com a aplicação.

Na implementação, imagine que temos um arquivo dedicado à gravação de dados. Nesse contexto, importamos o método `WriteFile` do módulo FS e preparamos um exemplo de conteúdo a ser escrito no arquivo. A chamada do método inclui o caminho desejado para o arquivo, o conteúdo especificado, e a função de callback para o caso de erro, proporcionando uma visualização clara da estrutura e execução.

Prosseguindo, discutiremos alguns detalhes cruciais que podem surgir durante a implementação dessa funcionalidade na sua aplicação. Acessando o código através do VS Code, é importante estar atento as particularidades do JavaScript, como a gestão de imports e a validação de modelos, que podem influenciar diretamente no sucesso da operação de gravação.

Um aspecto a ser destacado é a importância de garantir a correta importação dos modelos necessários, como `patient` e `doctor`, no seu código, para assegurar que as validações e buscas funcionem como esperado. Além disso, a atenção ao modelo de `prescription` é vital, especialmente em relação à obrigatoriedade da data da prescrição, que deve ser flexível para refletir situações reais.

Aplicando na prática, utilizaremos o pacote `PDFKit` para gerar arquivos PDF, como exemplo de um módulo específico para um tipo de arquivo. Esse pacote nos permitirá criar arquivos de prescrição médica, ilustrando o processo de gravação com um contexto prático e relevante.

Ao prepararmos nossa aplicação para gerar o arquivo PDF, criaremos uma estrutura que reflete a informação essencial de uma prescrição, como dados do paciente, do médico, medicamentos prescritos, dosagem e instruções. Essa preparação envolve a criação de um documento PDF, definição de fontes, tamanhos de texto, e a montagem do conteúdo a ser incluído, seguindo as especificações do modelo de `prescription`.

Finalmente, com o arquivo gerado, enfatizaremos a importância de encerrar corretamente o documento para garantir a integridade dos dados gravados. Isso é crucial para qualquer tipo de arquivo manipulado, seja PDF, TXT, XML, ou CSV.

No próximo tópico, avançaremos para temas como upload e leitura de arquivos, complementando nosso conhecimento sobre a manipulação de arquivos em aplicações.

**Upload de anexos**

Neste segmento, nosso foco será o uso do `multer`, um módulo do Node.js projetado para o gerenciamento de storage e destino de arquivos carregados, frequentemente utilizado em conjunto com o Express para simplificar a manipulação de arquivos enviados. A função de upload de anexos revela-se particularmente útil em funcionalidades como perfis de usuários, onde a inclusão de fotos de perfil é comum. O `multer` emerge como uma solução eficaz para esse desafio, embora a escolha do módulo apropriado deva sempre considerar os requisitos específicos de seu projeto.

A utilização do `multer` pode ser ilustrada com uma configuração básica para o upload de um arquivo em uma aplicação. A implementação envolve a importação do `multer`, a definição do diretório de destino dos uploads, e a manipulação dos parâmetros de nome e tamanho do arquivo na requisição. Esse cenário nos permite observar as nuances do processo de upload, desde a configuração inicial até a resposta à requisição.

Adentrando o contexto de um sistema de consultório médico, a funcionalidade de upload de anexos adquire relevância na atualização de prescrições médicas, históricos de pacientes ou inclusão de exames antigos e novos no registro do paciente. Esse procedimento não apenas enriquece o histórico de consultas, mas também contribui para uma gestão eficiente de informações médicas.

Para integrar essa funcionalidade de upload em nossa aplicação, devemos seguir o padrão de desenvolvimento já estabelecido: revisão do modelo para incluir novos atributos necessários, adaptação do repositório para suportar alterações no modelo, preparação do serviço para processar a requisição do controlador e, finalmente, a configuração do controlador para gerenciar a interação com o usuário.

Ao abordarmos a prescrição médica, por exemplo, adicionaremos um atributo para armazenar o arquivo, que será definido como um tipo string e não obrigatório, refletindo a possibilidade de uma prescrição existir sem um arquivo associado. A atualização do repositório para incorporar o upload de arquivos e a modificação do serviço para aceitar o arquivo como um parâmetro são etapas subsequentes essenciais para a funcionalidade desejada.

A culminância desse processo ocorre no controlador, onde o mapeamento da rota de upload é estabelecido. A instalação e configuração do `multer` são cruciais para definir o storage e a nomenclatura dos arquivos, facilitando a organização e o acesso aos arquivos carregados. A rota de upload configurada no controlador determina como os arquivos serão recebidos e processados, garantindo que os arquivos sejam armazenados corretamente e associados às prescrições médicas pertinentes.

Ao implementar e testar a funcionalidade de upload de arquivos, é vital verificar a consistência dos imports em toda a aplicação e assegurar que todas as dependências estejam corretamente instaladas. Além disso, a

presença de autenticação em nossa aplicação requer a inclusão do token de autorização nas chamadas de teste, um detalhe crucial para a segurança e integridade do sistema.

Na continuação, exploraremos a leitura de arquivos, complementando nosso conhecimento sobre a manipulação de dados e arquivos em aplicações web.

**Leitura de arquivos**

Concluindo, nesta última parte, focaremos em adaptar nosso projeto para a leitura de arquivos, um elemento fundamental na manipulação de dados em uma aplicação.

Utilizaremos o módulo nativo `fs` (File System), já introduzido nas nossas discussões anteriores. Este tópico é essencial por sua simplicidade e eficiência na leitura de arquivos, similar ao método `writeFile` que exploramos previamente. Através do método `readFile` do módulo `fs`, podemos facilmente acessar o conteúdo de um arquivo, especificando o caminho do arquivo desejado e utilizando uma função de callback para manipular os dados lidos ou tratar eventuais erros.

Em nossa prática, a funcionalidade de leitura de arquivos será integrada ao projeto com a utilização adicional dos módulos `process` e `path`. Esses módulos são cruciais para o processamento de arquivos e o manejo de diretórios, permitindo-nos localizar e acessar arquivos eficazmente. Enquanto o `path` oferece ferramentas para o tratamento de caminhos de diretório, o `process` auxilia na obtenção de informações sobre o ambiente de execução do nosso projeto.

Prosseguindo para a implementação no controlador de prescrição, necessitaremos dos pacotes `process` e `path` para facilitar o acesso e a leitura dos arquivos relacionados às prescrições médicas. Esse procedimento alinha-se ao nosso objetivo de aproximar o projeto da

realidade, possibilitando, por exemplo, a recuperação de exames ou prescrições médicas armazenados.

Na prática, realizaremos a leitura de um arquivo específico associado a uma prescrição médica. Para isso, importaremos os pacotes mencionados e utilizaremos a função `resolve` do módulo `path`, juntamente com `process.cwd()`, para construir o caminho até o arquivo desejado. Esse processo será executado em uma estrutura de rota `GET`, refletindo a operação de leitura na nossa API.

A implementação incluirá a busca por uma prescrição específica por meio de seu ID, seguida pela construção do caminho do arquivo e sua leitura efetiva. O resultado dessa operação será o caminho do arquivo lido, que será então retornado como resposta à requisição. Esse método nos permite não apenas acessar os dados de uma prescrição específica, mas também recuperar e disponibilizar o arquivo relacionado, enriquecendo a funcionalidade do nosso sistema.

À medida que avançamos para o encerramento, enfatizo a importância de testar e validar a implementação dessa funcionalidade. As próximas etapas do nosso curso incluirão o desenvolvimento do frontend e a realização de testes, permitindo uma compreensão abrangente tanto do backend quanto das interações do usuário com o sistema.

Espero que estejam tão entusiasmados quanto eu para continuar evoluindo em nosso projeto. Nos próximos encontros, ampliaremos nossos conhecimentos e habilidades, preparando-os não apenas para desafios acadêmicos, mas também para oportunidades profissionais no mundo do desenvolvimento de software. Até a próxima aula, onde exploraremos novas fronteiras em nossa jornada de aprendizado.

**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**

Para complementar os conhecimentos adquiridos em nossa aula, recomendo um aprofundamento prático e teórico através do vídeo "Manipulando arquivos e pastas com Java", produzido pelo canal Dev Superior, disponível no YouTube. Este material didático gratuito proporciona uma visão detalhada sobre como realizar operações de leitura, escrita, criação e gerenciamento de arquivos e diretórios usando a linguagem Java, uma habilidade essencial para qualquer desenvolvedor.

**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 Oracle11g. Pearson: 2013.

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

**Ir para exercício**