UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
DEPARTAMENTO DE COMPUTAÇÃO
IC599 - SISTEMAS WEB I

CAIO MARINHO DOS REIS - 20240035240
Maria Fernanda Galdino de Oliveira - 20240016020

PRIMEIRA AVALIAÇÃO

Seropédica
2025

PARTE 1

O início da primeira parte do trabalho consistiu na escolha de substituição do conteúdo Confetti Cuisine para o tema desejado pela dupla: um site informativo sobre a artista coreana Jennie e sobre o último álbum lançado pela mesma - *Ruby*.

No começo, informações contidas no arquivo **index.html** - página principal - foram modificadas para assim então dar prosseguimento às seguintes páginas. Imagens relacionadas ao tema de projeto foram adicionadas à pasta **images** e o **nodemon** foi instalado para que o servidor reinicie automaticamente após cada alteração, tornando o processo de desenvolvimento mais dinâmico.

Gets do arquivo main.js foram modificados para a utilização das novas imagens relacionadas ao tema escolhido. Nesta etapa a dupla encontrou dificuldades para as imagens aparecerem no site, o que demandou análise do código presente no arquivo a fim de compreender as requisições e a função getfile presente no arquivo utils para buscar a imagem e colocá-la no site. A fim de não modificar a lógica do arquivo fornecido, optou-se por manter os nomes - mesmo que os conteúdos tenham sido modificados - e as imagens já presentes no projeto, utilizando dos gets para buscar novas imagens.

PARTE 2

A pasta **public** contém:

- A subpasta **css**, com as estilizações das páginas;
- A subpasta **images**, com as imagens utilizadas no site.

A pasta **views** contém as páginas e seus respectivos arquivos HTML.

- **O arquivo main**

O código no arquivo main cria um servidor. O servidor responde a diferentes requisições HTTP, como abrir páginas HTML, carregar imagens, arquivos CSS e JavaScript. O servidor é criado com o módulo nativo http e configurado para ouvir na porta 3000.

O código utiliza módulos personalizados para organizar: Rotas, Tipos de conteúdo e Funções auxiliares.

São definidos caminhos para responder a diferentes URLs, como:

/ → Página inicial (index.html)
/courses.html, /contact.html → Outras páginas

Arquivos de imagem, CSS e JS também são tratados por suas respectivas rotas. Cada rota envia uma resposta com: Um código de status HTTP (como 200 para sucesso), um tipo de conteúdo apropriado (HTML, CSS, imagem etc.) e o conteúdo do arquivo solicitado (usando utils.getFile).

- **Arquivo content types**

Este trecho de código JavaScript define um módulo que exporta um objeto contendo mapeamentos de tipos de arquivo para seus respectivos Content-Type headers, que são usados em respostas HTTP para indicar o tipo de conteúdo que está sendo enviado pelo servidor. module.exports = { ... } - Esse comando exporta um objeto para que ele possa ser usado em outros arquivos do Node.js. Ou seja, quando outro arquivo fizer require deste módulo, ele receberá esse objeto.
Esse objeto pode ser usado para definir o cabeçalho correto de acordo com o tipo de arquivo que está sendo enviado ao cliente. Isso garante que o navegador saiba como tratar o arquivo recebido.

- **Arquivos package.json**

Este arquivo serve para registrar exatamente quais versões das dependências foram instaladas, garantindo que todos que instalarem o projeto tenham o mesmo ambiente, ajudando na reprodutibilidade e segurança do projeto. Nessa parte de código, constam nome e versão do projeto, tal como suas dependências.

- **Arquivo router**

```
const httpStatus = require("http-status-codes"),
 contentTypes = require("./contentTypes"),
 utils = require("./utils");
```

httpStatus: Importa um módulo para lidar com códigos de status HTTP (como 200, 404, etc.).

ContentTypes: Importa um módulo próprio que provavelmente define os tipos de conteúdo (Content-Type) para as respostas HTTP.

utils: Importa funções utilitárias, como para ler arquivos.

```
const routes = {
  GET: {},
  POST: {}
};
```

Cria um objeto routes que armazena funções (handlers) para cada rota, separadas por método HTTP (GET e POST).

```
exports.handle = (req, res) => {
  try {
    routes[req.method][req.url](req, res);
  } catch (e) {
    res.writeHead(httpStatus.OK, contentTypes.html);
    utils.getFile("views/error.html", res);
  }
};
```

SE DER ERRO

res.writeHead(...): escreve o cabeçalho da resposta, indicando sucesso e que o conteúdo será HTML.

utils.getFile("views/error.html", res);: envia o conteúdo do arquivo **error.html** como resposta. Isso mostra uma página de erro amigável.

Esse código tenta chamar uma rota conforme o método e a URL da requisição. Se algo der errado (ex: rota não existe), ele responde com uma página de erro HTML.

```
exports.get = (url, action) => {
  routes["GET"][url] = action;
```

```
};

exports.post = (url, action) => {
  routes["POST"][url] = action;
};
```

get(url, action): Cadastra uma função (handler) para uma rota GET específica.

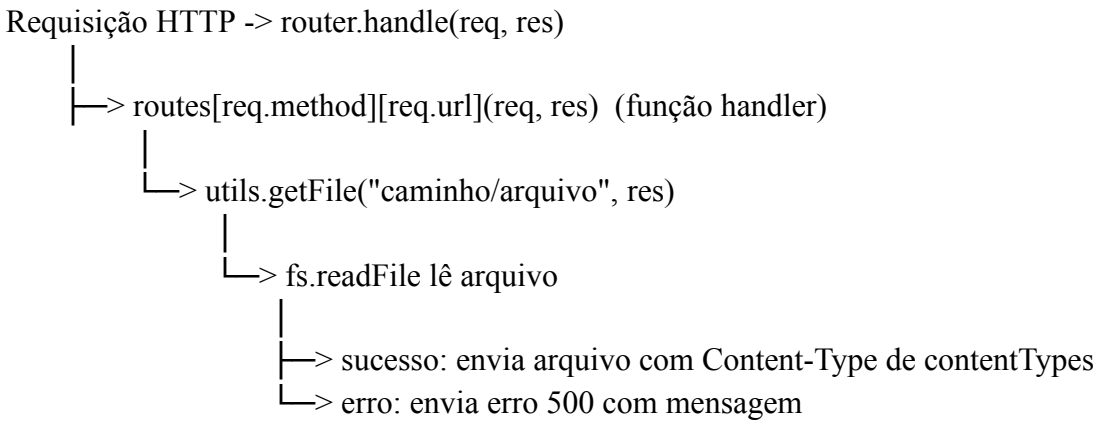
post(url, action): Cadastra uma função (handler) para uma rota POST específica.

Esse código é um roteador básico para um servidor Node.js. Ele permite registrar funções para rotas específicas (GET e POST). Quando chega uma requisição, ele executa a função correspondente e se não existir rota, mostra uma página de erro.

- **Arquivo utils**

Essa função é útil para servir arquivos estáticos (como páginas HTML, CSS, imagens) em um servidor HTTP simples. Ela abstrai a leitura do arquivo e o envio da resposta, tratando erros de forma adequada.

Como os arquivos interagem entre eles?



O arquivo que define os tipos de conteúdo (contentTypes.js) exporta um objeto com os cabeçalhos HTTP Content-Type para diferentes extensões de arquivo, garantindo que as respostas sejam enviadas com o tipo correto para o navegador interpretar. O router.js importa esse módulo para usar esses tipos de conteúdo e também importa o módulo de utilitários (utils.js), que contém a função getFile responsável por ler arquivos do sistema e enviar seu conteúdo na

resposta HTTP, tratando erros caso o arquivo não seja encontrado ou não possa ser lido. Quando uma requisição chega, o roteador verifica se existe uma função handler cadastrada para o método HTTP (GET ou POST) e a URL requisitada; se existir, executa essa função, que normalmente usará utils.getFile para enviar arquivos estáticos, como páginas HTML. Se não existir um handler para a rota, o roteador usa utils.getFile para enviar uma página de erro padrão.

Assim, o fluxo é: o servidor recebe a requisição, o roteador identifica a função correta para tratar essa rota, essa função pode chamar getFile para ler e enviar arquivos, e o cabeçalho Content-Type usado na resposta vem do módulo contentTypes, garantindo que o navegador entenda o tipo do conteúdo enviado. Essa interação modularizada permite que o servidor seja organizado, reutilizável e fácil de manter, respondendo adequadamente a diferentes requisições HTTP com os conteúdos corretos e tratamento de erros.