UNIVERSIDADE FEDERAL RURAL DO RIO DE JANEIRO
DEPARTAMENTO DE COMPUTAÇÃO

IC599 – Sistemas WEB I

Seropédica, março de 2025 Prof. Claver Pari Soto

Este conteúdo é baseado na tradução livre do livro: Get Programming with Node.js, autoria de Jonathan Wexler. 2019. Manning Publications

Criando Uma Primeira Aplicação Web Completa

Nas seguintes aulas será criada uma página de receitas e um espaço onde possíveis alunos possam se inscrever. Uma aplicação completa com múltiplas páginas. A aplicação tem três visualizações (*views*), rotas para essas visualizações e recursos, além de uma pasta pública para os arquivos do cliente. Para começar, será necessário desenvolver a lógica da aplicação com o objetivo de manter o código limpo e sem repetições. Depois, serão adicionadas algumas visualizações voltadas para o público e estilos personalizados. Na primeira etapa, teremos um servidor web capaz de lidar com requisições para arquivos e recursos específicos do projeto. O produto final será algo que poderá ser expandido gradualmente e conectar a um banco de dados, conforme o cliente solicitar.

Para criar esta aplicação, seguiremos os seguintes passos:

- Inicializar o arquivo `package.json` da aplicação.
- Configurar a estrutura de diretórios do projeto.
- Criar a lógica da aplicação no arquivo `main.js`.
- Criar três visualizações (*views*), cada uma contendo uma imagem clicável que possa ser servida de forma independente:
    - Index (página inicial)
    - Cursos
    - Contato
    - Obrigado
    - Erro
- Adicionar recursos personalizados (*assets*).
- Construir o roteador (*router*) da aplicação.
- Tratar os erros da aplicação.
- Executar a aplicação.

Inicializando a aplicação

Para começar, devemos usar o `npm` para criar um arquivo `package.json` com um resumo da aplicação a ser desenvolvida. Navegar até um diretório no computador onde será salvo este projeto e então criar uma nova pasta para o projeto. Nessa pasta executar: **`npm init`**.

Siguir as instruções na linha de comando e aceitar todos os valores padrão. Em seguida, instalar o pacote **http-status-codes** executando **npm install http-status-codes --save** na janela do terminal do projeto.

Estrutura de diretórios da aplicação

Na estrutura do projeto, os arquivos **main.js**, **package.json** e **router.js** devem ficar no nível raiz do meu diretório. Qualquer conteúdo HTML será representado por arquivos **.html** individuais, que ficarão dentro de uma pasta chamada **views** dentro da pasta do projeto.

O diretório completo do projeto da aplicação terá a seguinte estrutura:

```
.
|____main.js
|____router.js
|____public
| |____css
| | |____confetti_cuisine.css
| | |____bootstrap.css
| |____images
| | |____product.jpg
| | |____graph.png
| | |____cat.jpg
| | |____people.jpg
| |____js
| | |____confettiCuisine.js
|____package-lock.json
|____package.json
|____contentTypes.js
|____utils.js
|____views
| |____index.html
| |____contact.html
| |____courses.html
| |____thanks.html
| |____error.html
```

O servidor da aplicação responderá com arquivos HTML que estão na pasta views. Os recursos dos quais esses arquivos dependem ficarão em uma pasta chamada **public**.

> OBSERVAÇÃO: Os arquivos HTML serão visualizados pelo cliente, mas não são considerados recursos (assets) e não devem ser colocados na pasta **public**.

A pasta **public** conterá pastas chamadas **images**, **js** e **css** para armazenar os recursos voltados para o cliente da aplicação. Esses arquivos definem os estilos e as interações em JavaScript entre a aplicação e o usuário.

Para adicionar rapidamente um pouco de estilo na aplicação, foi adicionado o arquivo **bootstrap.css** baixado de http://getbootstrap.com/docs/4.0/getting-started/download/ e na pasta css dentro de **public**. Também foi criado um arquivo chamado **confetti_cuisine.css** para qualquer regra de estilo personalizada que se queira aplicar a este projeto.

Em seguida, deve se configurar a lógica da aplicação.

Criando main.js e router.js

Agora que está configurada a estrutura de pastas e inicializado o projeto, é preciso adicionar a lógica principal da aplicação ao site para que ele comece a servir arquivos na porta **3000**. Vamos manter as rotas em um arquivo separado, então é preciso importar esse arquivo junto com o módulo **fs** para poder servir arquivos estáticos.

Devemos criar um novo arquivo chamado **main.js**. Dentro desse arquivo, definir o número da porta da aplicação, importar os módulos **http** e **http-status-codes** e também os módulos personalizados que ainda serão criados: **router**, **contentTypes** e **utils**, como mostrado no trecho de código seguinte:

> OBSERVAÇÃO: Os módulos **contentTypes** e **utils** simplesmente ajudam a organizar as variáveis dentro do **main.js**.

Código: Conteúdo de **main.js** com os módulos requeridos

```
const port = 3000,
  http = require("http"),
  httpStatus = require("http-status-codes"),
  router = require("./router"),
  contentTypes = require("./contentTypes"),
  utils = require("./utils");
```

A aplicação não será iniciada até que seja criada os módulos locais, então deve se criar o **contentTypes.js**, usando o código da próxima listagem. Neste arquivo, estamos exportando um objeto que mapeia tipos de arquivos para seus respectivos valores de cabeçalho, para uso nas respostas. Mais tarde, será acessado o tipo de conteúdo HTML no **main.js** usando **contentTypes.html**.

Código: Mapeamento de objeto em contentTypes.js

```
module.exports = {
  html: {
    "Content-Type": "text/html"
  },
  text: {
    "Content-Type": "text/plain"
  },
  js: {
    "Content-Type": "text/js"
  },
  jpg: {
    "Content-Type": "image/jpg"
  },
  png: {
    "Content-Type": "image/png"
  },
  css: {
    "Content-Type": "text/css"
  }
};
```

Em seguida, deve se configurar a função a ser usada para ler o conteúdo dos arquivos em um novo módulo chamado **utils**. Dentro de **utils.js**, adicionar o código da próxima

listagem. Neste módulo, é necessário exportar um objeto que contém uma função chamada **getFile**. Essa função procura um arquivo no caminho fornecido. Se o arquivo não existir, é retornada imediatamente uma página de erro.

Código: Funções utilitárias em **utils.js**

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

module.exports = {
  getFile: (file, res) => {
    fs.readFile(`./${file}`, (error, data) => {
      if (error) {
        res.writeHead(httpStatus.INTERNAL_SERVER_ERROR, contentTypes.html);
        res.end("There was an error serving content!");
      }
      res.end(data);
    });
  }
};
```

Por fim, em um novo arquivo, adicionar o código da listagem a seguir. Esse arquivo **router.js** importa o módulo **http-status-codes** e os dois módulos personalizados: **contentTypes** e **utils**.

O módulo **router** inclui um objeto **routes** que armazena pares chave-valor mapeados para requisições GET por meio da função **get**, e para requisições POST por meio da função **post**. A função **handle** é aquela usada como função de retorno (*callback*) para o **createServer** no **main.js**.

As funções **get** e **post** recebem uma URL e uma função de **callback**, e então as mapeiam entre si dentro do objeto **routes**. Se nenhuma rota for encontrada, é usada a função personalizada **getFile** do módulo **utils** para responder com uma página de erro.

Código: Tratamento de rotas em **router.js**

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

const routes = {
  GET: {},
  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);
  }
};

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

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

Para fazer o servidor de aplicação funcionar, é preciso configurar as rotas e as visualizações (views) da aplicação.

Criando visualizações (*views*)

As visualizações são voltadas para o cliente e podem determinar o sucesso ou fracasso da experiência do usuário com a aplicação. Usaremos um modelo semelhante para cada página a fim de reduzir a complexidade nesta aplicação.

O topo de cada página HTML deve conter uma estrutura básica de HTML, uma seção **<head>**, um link para a folha de estilo personalizada (que ainda será criada) e a navegação. A página inicial do site **Confetti Cuisine** se parecerá com a figura seguinte, com links para as três visualizações no canto superior.

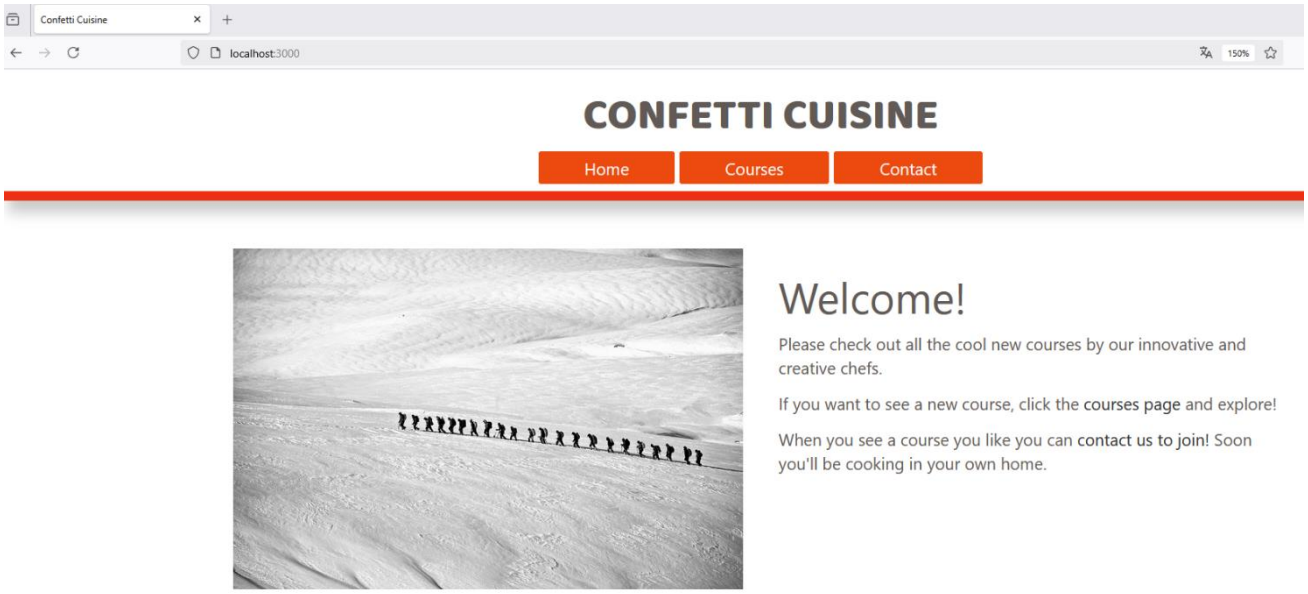


Figura. Exemplo de página inicial para a Confetti Cuisine

Para a página inicial, será criado uma nova visualização chamada **index.html** na pasta **views** e será adicionado o conteúdo específico da página inicial. Como estamos usando o **bootstrap.css**, precisamos fazer referência a esse arquivo nas páginas HTML adicionando **<link rel="stylesheet" href="/bootstrap.css">** dentro da tag **head** do HTML. Faremos o mesmo na folha de estilo personalizada, **confetti_cuisine.css**.

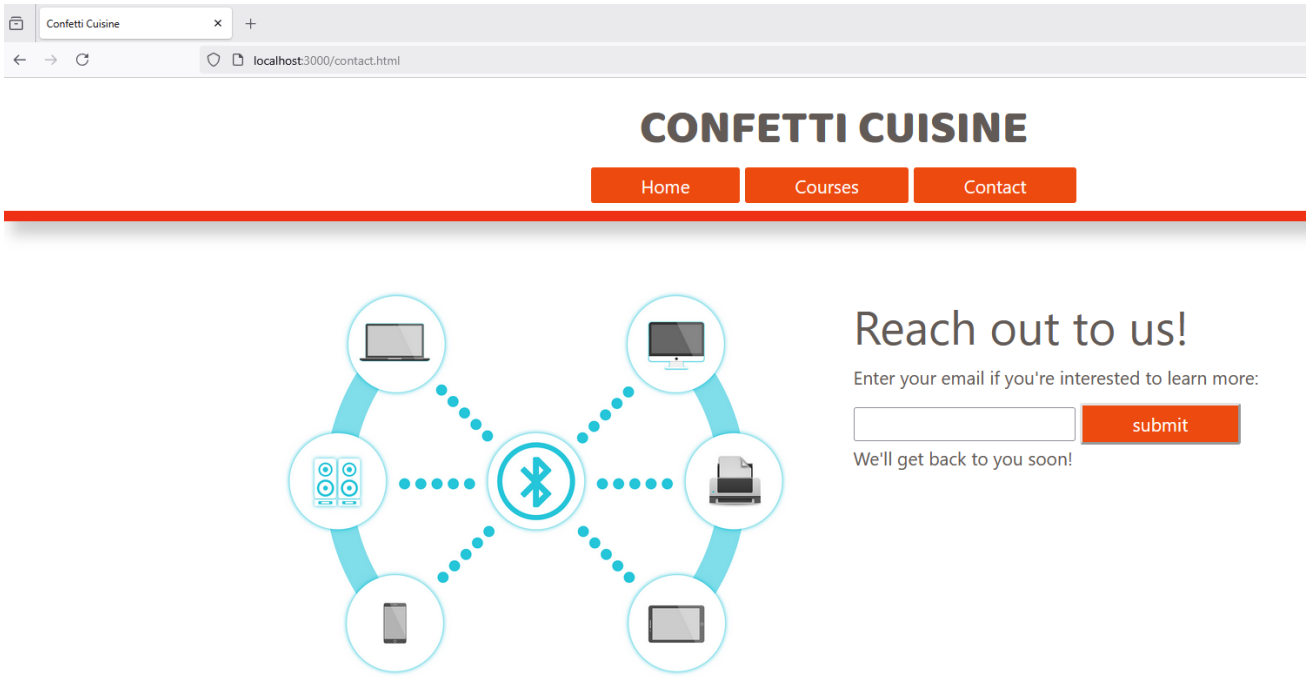
Em seguida, cria-se um arquivo **courses.html** para exibir uma lista das aulas de culinária disponíveis e um arquivo **contact.html** com o formulário a seguir. Esse formulário envia informações de contato via **POST** para a rota /. O código do formulário deve se parecer com o da próxima listagem.

Código: Exemplo de formulário que envia (**POST**) para a rota da página inicial em **contact.html**

```
<form class="contact-form" action="/" method="post">
  <input type="email" name="email" required />
```

```
        <input class="button" type="submit" value="submit" />
    </form>
```

A página de contato do site será parecida com a figura a seguir:



Exemplo de página de contato para a Confetti Cuisine

Cada página faz link para as outras por meio de uma barra de navegação. Precissamos garantir que todos os recursos que estamos usando nesses arquivos estejam corretamente definidos quando sejam criadas as rotas. Se algum recurso estiver faltando, a aplicação pode travar ao tentar procurar os arquivos correspondentes.

Vomos adicionar esses recursos para que as páginas tenham conteúdos mais ricos.

Adicionando recursos

Para esta aplicação, foram criados alguns estilos personalizados que serão usados por cada uma das visualizações. Quaisquer mudanças de cor, dimensão ou posicionamento que se queira fazer nos elementos do site serão colocadas no arquivo **`confetti_cuisine.css`**, que fica em **`public/css`**, ao lado do **`bootstrap.css`**.

Quando este arquivo for salvo, as visualizações terão cores e estrutura ao serem carregadas. Se for decidido usar algum JavaScript do lado do cliente, será preciso criar um arquivo **`.js`**, adicioná-lo à pasta **`public/js`** e fazer referência a ele dentro de cada arquivo usando as tags **`<script>`**.

Por fim, adicionaremos as imagens à pasta **`public/images`**. Os nomes dessas imagens devem corresponder aos nomes que sejam usados dentro das visualizações HTML.

O único passo restante é registrar e tratar as rotas para cada visualização e recurso no projeto.

Criando rotas

A última parte do quebra-cabeça é uma das mais importantes: as rotas. As rotas da aplicação determinarão quais URLs são acessíveis ao cliente e quais arquivos serão servidos.

Foi criado especificamente um arquivo **router.js** para gerenciar as rotas, mas ainda é preciso registrá-las. Registrar as rotas essencialmente significa passar uma URL e uma função de **callback** para as funções **router.get** ou **router.post**, dependendo de qual método HTTP estamos tratando. Essas funções adicionam as rotas ao objeto **router.routes**, que mapeia as URLs para as funções de callback a serem invocadas quando essa URL for acessada.

Recapitulando, para registrar uma rota, é preciso especificar o seguinte:

- Se a requisição é uma requisição GET ou POST
- O caminho da URL
- O nome do arquivo a ser retornado
- O código de status HTTP
- O tipo do arquivo sendo retornado (como o tipo de conteúdo)

Em cada função de callback, é preciso indicar o tipo de conteúdo que será enviado na resposta e usar o módulo **fs** para ler o conteúdo das visualizações e recursos na resposta. Adicionamos as rotas e o código na próxima listagem abaixo das linhas require no **main.js**.

Código: Registrando rotas individuais com o módulo **router** no **main.js**

```
router.get("/", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.html);
  utils.getFile("views/index.html", res);
});

router.get("/courses.html", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.html);
  utils.getFile("views/courses.html", res);
});

router.get("/contact.html", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.html);
  utils.getFile("views/contact.html", res);
});

router.post("/", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.html);
  utils.getFile("views/thanks.html", res);
});

router.get("/graph.png", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.png);
  utils.getFile("public/images/graph.png", res);
});

router.get("/people.jpg", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.jpg);
  utils.getFile("public/images/people.jpg", res);
});
```

```
router.get("/product.jpg", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.jpg);
  utils.getFile("public/images/product.jpg", res);
});

router.get("/confetti_cuisine.css", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.css);
  utils.getFile("public/css/confetti_cuisine.css", res);
});

router.get("/bootstrap.css", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.css);
  utils.getFile("public/css/bootstrap.css", res);
});

router.get("/confetti_cuisine.js", (req, res) => {
  res.writeHead(httpStatus.OK, contentTypes.js);
  utils.getFile("public/js/confetti_cuisine.js", res);
});

http.createServer(router.handle).listen(port);
console.log(`The server is listening on port number: ${port}`);
```

OBSERVAÇÃO: Observe a rota POST, que irá tratar os envios de formulários na página contact.html. Em vez de responder com outra página HTML, essa rota responde com uma página HTML de "obrigado por apoiar o produto".

Agora devemos iniciar a aplicação com node **`main.js`** e navegar até `http://localhost:3000` para ver a página inicial da aplicação web.

OBSERVAÇÃO: Só foram criadas rotas para os recursos (imagens, js e css) que estão representados como arquivos dentro do projeto.