Aula 4

Created	@January 13, 2024 8:30 PM
Data	@January 15, 2024
Status	Iniciado
Tipo	Aula

Mergulhando no Express (Setup e Organização)

Um panorama sobre a arquitetura MVC (Model View Controller)

Vamos usar uma analogia simples para explicar como funciona uma arquitetura MVC (Model-View-Controller) de uma forma fácil de entender:

Imagine que você está construindo uma casa:

1. **Modelo (Model):**
    - **Analogia:** Os modelos são como as plantas da casa.
    - **Explicação:** Eles definem a estrutura e a lógica da casa. Isso inclui quantos quartos, banheiros, e como eles se relacionam. O modelo gerencia os dados e a lógica de negócios da aplicação, assim como as plantas gerenciam o layout da casa.
2. **Visão (View):**

- **Analogia:** As vistas são como as janelas e a decoração da casa.
- **Explicação:** Elas representam como a casa (dados do modelo) é apresentada aos moradores. Cada janela (visão) mostra uma parte diferente da casa, e a decoração (visão) dá uma aparência agradável. As vistas são responsáveis por exibir as informações da aplicação de uma maneira compreensível para o usuário.

3. **Controlador (Controller):**
   - **Analogia:** Os controladores são como os arquitetos e empreiteiros da casa.
   - **Explicação:** Eles lidam com a comunicação entre as plantas (modelo) e a construção (visão). Se algo precisa ser modificado ou adicionado, os controladores fazem os ajustes necessários. Eles também gerenciam as interações dos moradores (usuários) com a casa, como abrir uma porta ou janela.

Em resumo, na arquitetura MVC:

- **Modelo (Model):** Gerencia a lógica de negócios e os dados.
- **Visão (View):** Apresenta os dados de maneira compreensível para o usuário.
- **Controlador (Controller):** Facilita a comunicação entre o modelo e a visão, lidando com as interações do usuário.

Esta abordagem divide a lógica da aplicação em partes separadas, tornando o desenvolvimento e a manutenção mais organizados e eficientes. Assim como uma casa é construída com diferentes especialidades trabalhando juntas, a arquitetura MVC divide a construção de uma aplicação em partes colaborativas e gerenciáveis.

Exemplo MVC em express

Vamos criar um exemplo simples de uma aplicação Express.js utilizando a arquitetura MVC. Neste exemplo, teremos um modelo para gerenciar dados, uma visão para exibir informações e um controlador para lidar com as solicitações do cliente.

1. **Instalação de Dependências:**
   Certifique-se de ter o Express.js instalado. Se não tiver, você pode instalá-lo com o

seguinte comando:

```
bashCopy code
npm install express
```

2. **Estrutura do Projeto:**
   Crie uma estrutura básica de projeto com pastas para modelos, visões e controladores. Por exemplo:

```
markdownCopy code
- projeto-mvc
  - controllers
    - mainController.js
  - models
    - dataModel.js
  - views
    - indexView.js
  - app.js
```

3. **Conteúdo dos Arquivos:**

   - **dataModel.js (Model):**

```
javascriptCopy code
// models/dataModel.js
class DataModel {
  constructor() {
    this.data = [];
  }

  getAllData() {
    return this.data;
  }
```

```
  addData(item) {
    this.data.push(item);
  }
}

module.exports = DataModel;
```

- **indexView.js (View):**

```
javascriptCopy code
// views/indexView.js
class IndexView {
  showData(data) {
    console.log('Dados:', data);
  }
}

module.exports = IndexView;
```

- **mainController.js (Controller):**

```
javascriptCopy code
// controllers/mainController.js
const express = require('express');
const DataModel = require('../models/dataModel');
const IndexView = require('../views/indexView');

const router = express.Router();
const model = new DataModel();
const view = new IndexView();

router.get('/', (req, res) => {
  const data = model.getAllData();
```

```
  view.showData(data);
  res.send('Dados exibidos no console. Verifique o cons
ole do servidor.');
});

router.post('/add', (req, res) => {
  const newItem = 'Novo Item';
  model.addData(newItem);
  res.send('Novo item adicionado. Verifique o console d
o servidor.');
});

module.exports = router;
```

- **app.js (Aplicação Express):**

```
javascriptCopy code
// app.js
const express = require('express');
const mainController = require('./controllers/mainContr
oller');

const app = express();
const port = 3000;

app.use('/', mainController);

app.listen(port, () => {
  console.log(`Servidor rodando em http://localhost:${p
ort}`);
});
```

4. **Execução da Aplicação:**
   Execute o servidor com o comando:

   ```
   bashCopy code
   node app.js
   ```

   Visite `http://localhost:3000` no navegador e veja como o modelo, a visão e o controlador estão colaborando para exibir e adicionar dados. Este é um exemplo básico, e em aplicações mais complexas, você poderia expandir essa estrutura de acordo com as necessidades específicas do seu projeto.

Trabalhando com variáveis de ambiente (dotenv / process.env)

As variáveis de ambiente são valores externos ao código-fonte de uma aplicação que podem ser acessados pelo código durante a execução. Elas são frequentemente usadas para armazenar configurações sensíveis ou valores que podem variar dependendo do ambiente em que a aplicação está sendo executada (desenvolvimento, produção, teste, etc.).

O `dotenv` é uma biblioteca Node.js que facilita o carregamento de variáveis de ambiente a partir de um arquivo chamado `.env`. Este arquivo é geralmente mantido fora do controle de versão para garantir que informações sensíveis, como chaves de API, senhas e outras configurações específicas do ambiente, não sejam compartilhadas publicamente.

Aqui estão os passos básicos para utilizar variáveis de ambiente com `dotenv` no Node.js:

1. **Instalação do `dotenv`:**
   Você precisa instalar a biblioteca `dotenv` no seu projeto usando o seguinte comando:

```
bashCopy code
npm install dotenv
```

2. **Criação do arquivo `.env`:**
   Crie um arquivo chamado `.env` no diretório raiz do seu projeto. Adicione variáveis de ambiente neste arquivo usando o formato `CHAVE=VALOR`. Por exemplo:

```
envCopy code
PORT=3000
API_KEY=suachaveapi123
```

   Essas variáveis podem ser acessadas no seu código como `process.env.PORT` e `process.env.API_KEY`.

3. **Configuração no Código:**
   No seu código Node.js, carregue as variáveis de ambiente utilizando o `dotenv`. No início do seu arquivo principal (por exemplo, `app.js`), adicione o seguinte código:

```
javascriptCopy code
require('dotenv').config();
```

   Isso carregará automaticamente as variáveis do arquivo `.env` no objeto `process.env` durante a inicialização do aplicativo.

4. **Uso das Variáveis de Ambiente:**
   Agora, você pode acessar as variáveis de ambiente no seu código da seguinte maneira:

```
javascriptCopy code
const port = process.env.PORT || 3000; // Usando uma porta padrão se a variável não estiver definida
```

```
const apiKey = process.env.API_KEY;
```

O uso de `process.env.VARIAVEL` permite que você acesse os valores das variáveis de ambiente no seu código.

Certifique-se de adicionar o arquivo `.env` ao seu arquivo `.gitignore` para evitar o rastreamento no controle de versão, especialmente se ele contiver informações sensíveis.

O uso de variáveis de ambiente e `dotenv` ajuda a manter a segurança e a flexibilidade da sua aplicação, permitindo que você ajuste facilmente as configurações específicas do ambiente sem alterar o código-fonte.

Gerando o scaffold do projeto com express-generator

O termo "scaffold" se refere a um conjunto de arquivos e diretórios que fornecem uma estrutura inicial para o desenvolvimento de um aplicativo. Um scaffold pode incluir esqueletos de código, arquivos de configuração e uma organização de diretórios que ajuda os desenvolvedores a começarem um projeto de maneira rápida e padronizada. Essa abordagem é particularmente útil em frameworks web, onde existe uma estrutura comum para muitos tipos de aplicativos.

O `express-generator` é uma ferramenta para o Express.js que cria um scaffold básico para um aplicativo web usando o Express. Ele ajuda a iniciar rapidamente um projeto Express com uma estrutura de diretórios organizada e um conjunto mínimo de arquivos essenciais. Isso é especialmente útil quando você deseja evitar a configuração inicial manual e começar a trabalhar em seu aplicativo de maneira eficiente.

Aqui está um guia básico de como usar o `express-generator` na prática:

1. **Instalação:**
   Certifique-se de ter o `express-generator` instalado globalmente. Se não tiver, você pode instalá-lo com o seguinte comando:

```
bashCopy code
npm install -g express-generator
```

2. **Criação de um Projeto Express:**
   Crie um novo projeto Express usando o `express-generator`. Substitua "nome-do-projeto" pelo nome desejado do seu projeto.

   ```
   bashCopy code
   express nome-do-projeto
   ```

   Isso criará uma estrutura básica de diretórios e arquivos em "nome-do-projeto".

3. **Instalação de Dependências e Inicialização do Projeto:**
   Acesse o diretório do projeto e instale as dependências usando:

   ```
   bashCopy code
   cd nome-do-projeto
   npm install
   ```

   Em seguida, inicie o servidor:

   ```
   bashCopy code
   npm start
   ```

   Isso iniciará o servidor Express na porta padrão (geralmente 3000). Você pode acessar o aplicativo em `http://localhost:3000` no seu navegador.

O scaffold gerado pelo `express-generator` incluirá uma estrutura de pastas comuns, middleware básico, arquivos de configuração e um aplicativo Express funcional. Isso proporciona uma base sólida para começar a desenvolver sua aplicação sem a necessidade de configurar tudo manualmente.

A utilidade do scaffold é que ele fornece uma estrutura consistente, seguindo as melhores práticas, e economiza tempo no início do desenvolvimento, permitindo que você se concentre diretamente na lógica do seu aplicativo em vez de configurar a estrutura inicial.

Prático:

- A partir do projeto da aula anterior, faça a separação das routes e views do projeto manualmente;
- Inicie um novo projeto, utilizando o express-generator;
- Ajuste a estrutura original, adicionando novas pastas que façam sentido para a organização (models / config / etc);
- Instale e configure o módulo dotenv para utilização de variáveis de ambiente;
- Instale e substitua a view engine PUG pela EJS;
- Ajuste as views, para que estas passem a receber dados a partir de arquivos estáticos em formato .json;