Unifacs

Ciência da computação

A3 da UC Sistemas Distribuídos e Mobile.

João Vitor Gradin Padin

(1272224769)

João Vitor da Silva de Almeida Domingues

(1272229434)

Pedro Lima Lopes

(1272018234)

Pedro Henrique Fraga Ribeiro Cruz

(12722212371)

Vitor Moreira dos Santos

(12722213065)

Captação de dados de venda de uma rede de lojas

Salvador Bahia

2023

Relatório detalhado sobre como o nosso código referente ao Projeto A3 de captação de dados de venda de uma rede de lojas.

Front-End:

Estrutura HTML: A estrutura básica da página é definida usando HTML. Isso inclui a criação de um formulário com campos para a peça de roupa, quantidade e preço. Há também um botão para enviar o formulário.

Manipulação de eventos: Quando o botão de envio é clicado, a função **submitForm()** é chamada. Esta função obtém os valores inseridos nos campos do formulário e tenta enviá-los para um servidor **back-end.**

Comunicação com o servidor: A função **submitForm()** usa a API Fetch para enviar uma solicitação POST para **'http://localhost:3001/api/sales'**. Os dados do formulário são enviados no corpo da solicitação em formato JSON.

Tratamento de erros: Se ocorrer um erro durante a tentativa de enviar os dados, ele será registrado no console do navegador.

Atualização dos dados de vendas: Após uma venda ser enviada com sucesso, a função **updateSalesData()** é chamada para atualizar a lista de vendas exibida na página. Esta função também usa a API Fetch para obter todas as vendas do servidor back-end.

Exibição dos dados de vendas: Os dados de vendas recebidos do servidor são usados para atualizar o conteúdo de um elemento **div** na página. Para cada venda, um novo parágrafo é adicionado ao **div**, contendo os detalhes da venda.

Inicialização: Quando a página é carregada, a função **updateSalesData()** é chamada imediatamente para exibir todas as vendas existentes.

Banco de Dados

Esta parte do código é um exemplo de uma aplicação web em Node.js que utiliza o framework Express e o MySQL como banco de dados. Vamos analisar cada parte do código:

Importação de módulos: O código começa importando os módulos necessários, que são o express e o mysql2.

Configuração do servidor: Em seguida, uma instância do Express é criada e a porta 3001 é definida para o servidor.

Configuração do mecanismo de visualização: O mecanismo de visualização EJS é configurado para renderizar as páginas da web.

Rota de exemplo: Uma rota GET para a página inicial ('/') é definida, que renderiza a página 'cliente' com uma mensagem.

Conexão com o banco de dados MySQL: Uma conexão com o banco de dados MySQL é estabelecida usando as credenciais fornecidas.

Configuração do CORS: O CORS é configurado para permitir a comunicação com o front-end.

Início do servidor: O servidor começa a escutar na porta especificada.

Rotas da API: Várias rotas da API são definidas para obter relatórios do banco de dados. Cada rota executa uma consulta SQL específica e retorna os resultados.

Função para executar consultas: Uma função assíncrona executeQuery é definida para executar consultas no banco de dados.

Rota de teste: Finalmente, uma rota GET para a página inicial (‘/’) é definida, que envia a mensagem ‘joao lindo’.

Arquivo Loja Virtual

Esta parte do código é um script de criação de banco de dados MySQL. Ele cria um esquema chamado mydb e várias tabelas dentro desse esquema. Aqui está uma descrição detalhada de cada tabela:

tipoproduto: Esta tabela tem três colunas - id, nome e created_at. A coluna id é a chave primária.

status: Esta tabela tem duas colunas - id e situacao. A coluna id é a chave primária.

produto: Esta tabela tem nove colunas, incluindo id, tamanho, codigo_produto, tipo, id_estoque, created_at, tipoproduto_id, status e status_id. As colunas id e status_id são chaves primárias. Além disso, esta tabela tem duas chaves estrangeiras - tipoproduto_id e status_id, que referenciam as tabelas tipoproduto e status, respectivamente.

estoque: Esta tabela tem cinco colunas - id, descricao, quantidade, created_at e produto_id. A coluna id é a chave primária e produto_id é uma chave estrangeira que referencia a tabela produto.

gerente: Esta tabela tem cinco colunas - id, nome, created_at, produto_id e estoque_id. As colunas id e produto_id são chaves primárias. Além disso, esta tabela tem duas chaves estrangeiras - produto_id e estoque_id, que referenciam as tabelas produto e estoque, respectivamente.

pedido: Esta tabela tem oito colunas - id, id_cliente, id_usuario, data_pedido, data_venda, status e id_pedido_produto. A coluna id é a chave primária.

Cada tabela é criada com o mecanismo de armazenamento InnoDB. Além disso, o script também desativa temporariamente as verificações de chave única e chave estrangeira para facilitar a criação das tabelas.

Arquivo package.json/Node js

Este arquivo foi utilizado para gerenciar as dependências e scripts do nosso projeto.
Primeiramente utilizamos:

A tag **name:** Este campo é usado para identificar o projeto. Sendo o nome do projeto : “seu-projeto”.

Na tag **version**: Este campo especifica a versão atual do projeto. Neste caso, a versão é “1.0.0”.

Description: Este campo é usado para fornecer uma descrição do seu projeto. Neste caso, a descrição é “Descrição do projeto”.

Main: Este campo especifica o ponto de entrada principal do projeto. Sendo o ponto

de entrada é “index.js”.

O código que você forneceu é um exemplo de um arquivo package.json em um projeto Node.js. Este arquivo é usado para gerenciar as dependências e scripts do seu projeto. Aqui está um relatório detalhado sobre como o arquivo foi criado:

Scripts: Este campo é usado para definir scripts que podem ser executados a partir da linha de comando. Neste caso, há um script chamado “start” que executa “index.js”.

Dependencies: Este campo lista todas as dependências do projeto, ou seja, outros pacotes dos quais o projeto depende. As dependências são especificadas com seus nomes e versões. Neste caso, as dependências incluem “express”, “ejs”, “mysql”, “mysql2”, “node” e “nodemon”.

Keywords: Este campo pode ser usado para fornecer palavras-chave que ajudam outras pessoas a encontrar o seu projeto. Neste caso, não há palavras-chave especificadas.

Author: Este campo é usado para identificar o autor do projeto. O campo autor está vazio.

License: Este campo é usado para especificar a licença sob a qual o projeto é disponibilizado. Sendo a licença é “ISC”.

Arquivo package-lock.json

Sendo uma parte crucial do nosso projeto Node.js. Ele é gerado automaticamente sempre que o npm modifica a árvore de dependências do projeto, como durante uma operação npm install. O propósito deste arquivo é descrever a árvore de dependências exata que foi gerada, para que futuras instalações possam gerar árvores de dependências idênticas, independentemente das atualizações de dependências intermediárias.

Vamos dar uma olhada mais de perto em algumas partes do arquivo:

O campo **name** é onde o nome do projeto é especificado. Neste caso, o projeto é chamado de “back-end”.

O campo **lockfileVersion** indica a versão do **package-lock.json.** Aqui, a versão é 3.

O campo **requires** é um booleano que especifica se as dependências devem ser instaladas. Aqui, o valor é **true**, o que significa que as dependências devem ser instaladas.

No campo **packages** é um objeto que contém informações sobre cada pacote na árvore de dependências do projeto.

- No objeto “” (um objeto vazio) contém as dependências do projeto, que incluem “express”, “mysql”, “mysql2”, “node” e “nodemon”.
- Cada objeto subsequente na lista de pacotes representa uma dependência do projeto. Cada objeto de dependência contém informações sobre a versão da dependência, a URL de onde a dependência foi baixada, a integridade da dependência (um hash SHA que pode ser usado para verificar a integridade dos dados baixados), e quaisquer dependências da dependência.

Nós temos aqui uma parte de um arquivo package-lock.json do nosso projeto Node.js, que é crucial para o gerenciamento de dependências. Este arquivo é gerado automaticamente sempre que o npm modifica a árvore de dependências do projeto, como durante uma operação npm install. Ele descreve a árvore de dependências exata que foi gerada, de forma que futuras instalações possam gerar árvores de dependências idênticas, independentemente das atualizações de dependências intermediárias.

Vamos examinar algumas partes do arquivo:

node_modules/braces: Este pacote, na versão 3.0.2, é uma dependência do projeto. Ele tem uma dependência chamada “fill-range” e é compatível com versões do Node.js 8 ou superiores.

node_modules/bytes: Este pacote, na versão 3.1.2, é outra dependência do projeto. Ele é compatível com versões do Node.js 0.8 ou superiores.

node_modules/call-bind: Este pacote, na versão 1.0.5, tem três dependências: “function-bind”, “get-intrinsic” e “set-function-length”. Ele também tem um campo de financiamento, que provavelmente é usado para apoiar o desenvolvimento do pacote.

node_modules/chokidar: Este pacote, na versão 3.5.3, tem várias dependências, incluindo “anymatch”, “braces”, “glob-parent”, “is-binary-path”, “is-glob”, “normalize-path” e “readdirp”. Ele também tem uma dependência opcional chamada “fsevents”. Além disso, ele tem um campo de financiamento, que provavelmente é usado para apoiar o desenvolvimento do pacote.

node_modules/concat-map: Este pacote, na versão 0.0.1, é uma dependência do projeto.

node_modules/content-disposition: Este pacote, na versão 0.5.4, tem uma dependência chamada “safe-buffer”. Ele é compatível com versões do Node.js 0.6 ou superiores.

node_modules/content-type: Este pacote, na versão 1.0.5, é uma dependência do projeto. Ele é compatível com versões do Node.js 0.6 ou superiores.

node_modules/cookie: Este pacote, na versão 0.5.0, é uma dependência do projeto. Ele é compatível com versões do Node.js 0.6 ou superiores.

node_modules/cookie-signature: Este pacote, na versão 1.0.6, é uma dependência do projeto.

node_modules/core-util-is: Este pacote, na versão 1.0.3, é uma dependência do projeto.

node_modules/debug: Este pacote, na versão 2.6.9, é uma dependência do projeto.