Sistema Digital de Gerenciamento de Restaurantes

Documentação de Desenvolvimento

29 de janeiro de 2024

Sistema Digital de Gerenciamento de Restaurantes

Goiânia, GO

Versão 1.0

29 de janeiro de 2024

Conteúdo

1 Introdução

Este documento tem como objetivo fornecer uma visão abrangente do desenvolvimento e da arquitetura do "Sistema Digital de Gerenciamento de Restaurantes". Este sistema é uma solução tecnológica destinada a monitorar e gerenciar as atividades de um restaurante em diferentes contextos e cenários. Em particular, esta documentação aborda aspectos cruciais da arquitetura do servidor, que utiliza a plataforma Node.js em conjunto com o ORM Prisma, bem como detalhes relacionados à implementação do frontend usando a biblioteca ReactJS.

A abordagem adotada para desenvolver este sistema reflete um compromisso com a eficiência, escalabilidade e usabilidade. Para alcançar esses objetivos, exploramos cuidadosamente as ferramentas e tecnologias disponíveis, garantindo uma arquitetura sólida do servidor e uma interface de usuário intuitiva.

No decorrer deste documento, é possível encontrar informações detalhadas sobre a arquitetura do servidor Node.js, com descrições dos componentes fundamentais e detalhes de configuração. Além disso, examinaremos em profundidade o desenvolvimento do frontend com ReactJS, incluindo aspectos de design e interface do usuário.

A documentação também abordará aspectos do processo de desenvolvimento, incluindo metodologia e estrutura de pastas do projeto. Esperamos que este documento sirva como um guia abrangente para aqueles envolvidos no desenvolvimento, manutenção e uso do sistema, fornecendo uma base sólida para a compreensão de sua arquitetura e funcionamento.

Portanto, explore este documento e aprofunde o conhecimento sobre o "Sistema Digital de Gerenciamento de Restaurantes". Que ele possa servir como uma valiosa fonte de informações e orientações em sua jornada com este sistema inovador.

2 Servidor Node.js

Esta seção se inicia com uma visão geral do sistema do servidor, seção 2.1. A arquitetura do servidor Node.js é o alicerce tecnológico do nosso sistema, proporcionando uma base sólida e eficiente para a gestão das atividades.

Em seguida, na seção 2.2, apresenta-se os componentes do servidor Node.js que sustenta o sistema, é essencial desmembrar seus principais componentes e entender o papel de cada um. Vamos examinar em detalhes as partes fundamentais que compõem a arquitetura do servidor, incluindo o ambiente de execução Node.js, o uso do ORM Prisma e outros componentes-chave.

A configuração do servidor, seção 2.3, desempenha um papel crucial no desempenho e na segurança do sistema. Nesta subseção, abordaremos as configurações específicas do servidor Node.js, incluindo detalhes técnicos, ajustes de segurança e parâmetros de otimização. Uma configuração adequada é essencial para garantir que nosso servidor atenda às demandas de desempenho e segurança do sistema.

Além dos tópicos mencionados acima, é importante abordar questões de segurança e autenticação relacionadas à arquitetura do servidor Node.js, o que é encontrado na seção 2.4. Isso pode incluir a maneira como os dados sensíveis são tratados, medidas de segurança implementadas e autenticação de usuários e sistemas. Certificar-se de que nosso servidor seja seguro é uma prioridade fundamental para o sucesso do sistema.

Outro aspecto importante da arquitetura do servidor é o gerenciamento de dados e a integração com o banco de dados MySQL, conforme seção 2.5. Esta seção pode explorar como os dados são armazenados, acessados e manipulados pelo servidor, bem como estratégias de cache e otimização de consultas. A eficácia do nosso sistema depende em grande parte da integridade e do desempenho dos dados.

Por fim, na seção 2.6, abordaremos a escalabilidade do servidor Node.js. Este tópico é crucial, especialmente se o nosso sistema estiver sujeito a crescimento futuro. Discutiremos estratégias e práticas recomendadas para dimensionar o servidor de forma eficiente, garantindo que ele possa lidar com um aumento de carga de trabalho e demanda sem comprometer o desempenho e a confiabilidade.

Ao explorar esses tópicos, teremos uma compreensão completa da arquitetura do servidor Node.js que sustenta nosso sistema. Isso nos permitirá tomar decisões informadas durante o desenvolvimento, manutenção e expansão do sistema.

2.1 Visão Geral

O sistema é uma solução tecnológica que tem como objetivo monitorar, registrar e gerenciar as atividades comerciais e gastronômicas em diversos cenários e contextos. Nesta seção, exploraremos uma visão geral do funcionamento e da estrutura geral do servidor. Compreender essa arquitetura é fundamental para avaliar o desempenho, escalabilidade e robustez do nosso sistema.

A base do sistema é um servidor Node.js altamente escalável e eficiente que lida com a coleta, armazenamento e processamento de dados relacionados. A arquitetura do servidor inclui:

- **Node.js:** A plataforma Node.js oferece uma execução de alto desempenho para o servidor, permitindo a manipulação eficiente de solicitações e ações em tempo real.
- **ORM Prisma:** O ORM Prisma é usado para facilitar o acesso ao banco de dados MySQL, simplificando a manipulação de dados e garantindo uma interação segura e eficaz com o banco de dados.
- **Segurança e Autenticação:** Mecanismos robustos de segurança e autenticação são implementados para proteger dados sensíveis e controlar o acesso ao sistema.

A escolha de tecnologias adequadas desempenha um papel fundamental na construção de sistemas eficazes e confiáveis. No presente contexto, a implementação de Node.js, Prisma e autenticação por JWT emerge como uma combinação estratégica e crucial. Node.js, como plataforma de servidor, oferece eficiência e escalabilidade, permitindo que nosso sistema atenda a uma grande quantidade de solicitações de maneira ágil e confiável. Sua natureza não bloqueante e orientada a eventos é particularmente vantajosa, garantindo que o sistema possa lidar com tarefas intensivas em tempo real.

Além disso, a adoção do ORM Prisma representa um passo significativo em direção à simplificação do acesso ao banco de dados. O Prisma oferece uma camada de abstração que minimiza a complexidade das consultas e transações de banco de dados, reduzindo o risco de erros e vulnerabilidades de segurança. Isso não apenas acelera o desenvolvimento, mas também garante a integridade dos dados, uma consideração crítica quando se trata de informações relacionadas ao meio ambiente e ao monitoramento das atividades.

Por fim, a autenticação por JWT é uma pedra angular na segurança do sistema. A capacidade de verificar a identidade dos usuários e garantir que apenas indivíduos autorizados acessem o sistema é essencial para proteger informações sensíveis e controlar o acesso aos recursos. Através da emissão e verificação de tokens JWT, podemos estabelecer um sistema seguro de autenticação e autorização, fornecendo tranquilidade aos usuários e mantendo a confidencialidade dos dados relacionados.

Essas escolhas tecnológicas representam uma base sólida para o sucesso do "Sistema Digital de Gerenciamento de Restaurantes"e contribuem para sua capacidade de enfrentar os desafios relacionados ao monitoramento e controle das atividades.

2.2 Componentes do Servidor

Esta seção fornecerá uma compreensão sólida de como os diferentes elementos interagem para fornecer funcionalidade e eficiência ao nosso sistema.

Os componentes do servidor Node.js são peças-chave que desempenham funções específicas no contexto da aplicação. A seguir são discutidos os componentes que estão sendo utilizados e qual é o papel de cada um deles.

1. **Express:** O framework Express.js é o núcleo do servidor Node.js. Ele fornece a estrutura fundamental para criar aplicativos web e APIs RESTful. Este aplicativo é o ponto de entrada para todas as solicitações HTTP e é responsável pelo roteamento e manipulação das solicitações.
2. **ORM Prisma:** Prisma é um ORM (*Object-Relational Mapping*) que simplifica o acesso ao banco de dados. Uma instância do Prisma é usada para interagir com o banco de dados, executar consultas e manipular dados.
3. **CORS:** O pacote 'cors' é usado para habilitar a política de mesma origem em solicitações HTTP. Ele permite que o servidor aceite solicitações de diferentes origens, o que é útil quando deseja-se permitir que outros domínios acessem a API.

4. ***Middleware* de Body Parsing:** O *middleware* 'body-parser' é usado para analisar dados no corpo das solicitações HTTP, seja em formato JSON ou URL codificada. Isso é essencial para processar dados enviados por clientes. As configurações para analisar os dados JSON e URL codificada são estabelecidas de acordo com os padrões de projeto.

5. ***Middleware* Passport:** Passport é um *middleware* para autenticação. Ele é usado para autenticar solicitações de clientes, garantindo que apenas usuários autorizados tenham acesso a recursos protegidos.

6. ***Middleware* de Roteamento:** O comando `app.use(router)` define o *middleware* de roteamento. Ele é responsável por definir como as solicitações HTTP são tratadas e direcionadas para as rotas corretas. No código, o 'router' representa as rotas definidas em outro arquivo.

7. **Tratamento de Erros:** Este *middleware* lida com erros não tratados. Quando ocorre um erro não esperado durante o processamento da solicitação, esse middleware é acionado. Ele registra o erro no console e retorna uma resposta de erro com um status HTTP 500.

8. ***Upload* de arquivos** A biblioteca `path` é usada para lidar com caminhos de arquivos e diretórios. O `multer` é uma biblioteca Node.js que simplifica o processamento de formulários do tipo `multipart/form-data`, geralmente usados para upload de arquivos. A biblioteca `cripto` faz parte do conjunto de módulos nativos do Node.js e é usada para gerar um textithash (código de verificação) para o nome do arquivo.

2.2.1 Node.js

O Node.js é um ambiente de tempo de execução JavaScript de código aberto e multiplataforma. Ele é uma ferramenta popular para quase qualquer tipo de projeto. O Node.js utiliza o motor JavaScript V8, que é o núcleo do Google Chrome, fora do ambiente do navegador. Isso permite que o Node.js seja altamente eficiente em termos de desempenho.

Uma aplicação Node.js é executada em um único processo, sem criar uma nova textitthread para cada solicitação. O Node.js oferece um conjunto de primitivas de I/O assíncronas em sua biblioteca padrão, o que impede que o código JavaScript bloqueie. Em geral, as bibliotecas no Node.js são escritas usando paradigmas não bloqueantes, tornando o comportamento de bloqueio a exceção, em vez da regra.

Quando o Node.js executa uma operação de I/O, como leitura de rede, acesso a um banco de dados ou ao sistema de arquivos, em vez de bloquear a textitthread e gastar ciclos de CPU esperando, o Node.js retomará as operações quando a resposta chegar. Isso permite que o Node.js lide com milhares de conexões simultâneas com um único servidor, sem introduzir a complexidade de gerenciamento de concorrência de textitthreads, que poderia ser uma fonte significativa de bugs.

O Node.js tem uma vantagem única, pois milhões de desenvolvedores textitfront-end que escrevem JavaScript para o navegador agora podem escrever o código do lado do servidor, além do código do lado do cliente, sem a necessidade de aprender uma linguagem completamente diferente. No Node.js, os novos padrões ECMAScript podem ser usados sem problemas, pois, não é necessário esperar que todos os usuários atualizem seus navegadores - é possível decidir qual versão do ECMAScript usar alterando a versão do Node.js e também pode-se habilitar recursos experimentais específicos executando o Node.js com flags. Isso oferece uma flexibilidade valiosa para os desenvolvedores.

2.2.2 ORM Prisma

O Prisma é um ORM de última geração que tem o código aberto. Ele é composto pelas seguintes partes: um construtor de consultas auto gerado e seguro para tipos, projetado para Node.js e TypeScript, conhecido como *Prisma Client*; um sistema de migração que facilita a evolução do esquema do banco de dados ao longo do tempo, conhecido como *Prisma Migrate*; e o *Prisma Studio*, que é uma interface gráfica do usuário que permite visualizar e editar dados em seu banco de dados.

Dentre essas partes, o Prisma Studio é a única que não é de código aberto, o que significa que só pode ser executado localmente.

O Prisma Client pode ser utilizado em qualquer aplicação de backend Node.js (nas versões suportadas) ou TypeScript, incluindo aplicações serverless e microservices. Isso abrange uma ampla variedade

de tipos de aplicativos, desde APIs REST, APIs GraphQL e APIs gRPC até qualquer outra aplicação que necessite de um banco de dados.

O Prisma é uma ferramenta poderosa que simplifica o acesso e a manipulação de bancos de dados, fornecendo uma camada de abstração intuitiva para os desenvolvedores. Ele se destaca por sua facilidade de uso e integração com tecnologias modernas, tornando-o uma escolha valiosa para aqueles que desejam desenvolver aplicativos eficientes e robustos com facilidade.

2.3 Configuração do Servidor

A configuração do servidor desempenha um papel fundamental no sucesso de qualquer projeto, especialmente aqueles que envolvem tecnologias como Node.js. Para garantir que o servidor atenda aos requisitos de desempenho e segurança do sistema, é importante considerar uma configuração ideal.

Em relação às configurações técnicas, foram escolhidas as versões adequadas das bibliotecas e dependências do Node.js, garantindo as versões mais estáveis e seguras. Além disso, a alocação de recursos, como memória e CPU, deve ser cuidadosamente ajustada para atender às necessidades específicas do aplicativo. Isso inclui a configuração de limites de requisições e conexões para evitar sobrecarga do servidor.

A segurança é uma prioridade crítica, e a configuração deve incluir práticas recomendadas, como o uso de HTTPS para comunicação segura, a implementação de proteção contra ameaças conhecidas, como injeção de SQL e ataques de negação de serviço (DDoS), e a aplicação de políticas rigorosas de gerenciamento de chaves para garantir a integridade dos dados.

Além disso, otimizações de desempenho, como o uso de cache e compactação de recursos estáticos, podem ser implementadas para melhorar a velocidade de resposta do servidor. Monitoramento e registros detalhados também devem ser configurados para que se possa rastrear o desempenho e a segurança do sistema em tempo real.

Testar a configuração do servidor envolve a criação de testes automatizados que avaliam se as configurações técnicas, de segurança e de desempenho estão corretamente implementadas. Esses testes são essenciais para identificar e corrigir problemas potenciais antes que o sistema entre em produção. Além disso, os testes podem verificar se as configurações de segurança estão eficazes, identificando vulnerabilidades comuns, como brechas de segurança ou configurações inadequadas que possam expor o sistema a ameaças externas.

2.4 Segurança e Autenticação

A segurança é uma preocupação central no desenvolvimento de sistemas de software. À medida que mais informações pessoais e confidenciais são transferidas e armazenadas digitalmente, a necessidade de proteger esses dados se torna cada vez mais crítica. A autenticação é um dos principais pilares da segurança em sistemas modernos, e o uso de tecnologias como JSON Web Tokens (JWT) e o framework Passport desempenha um papel fundamental nesse contexto.

O JSON Web Token (JWT) é um padrão aberto (RFC 7519) que define um método compacto e autossuficiente para representar informações entre duas partes de uma forma que possa ser verificada e validada. Os JWTs são frequentemente usados para autenticação e autorização em aplicativos da web e serviços. Eles são compostos por três partes: um cabeçalho, uma carga útil (payload) e uma assinatura. O cabeçalho contém informações sobre como o JWT deve ser verificado, a carga útil contém os dados do usuário e a assinatura é usada para verificar a integridade do token.

A principal vantagem dos JWTs é que eles são autocontidos, o que significa que todas as informações necessárias para verificar a autenticidade de um token estão contidas nele mesmo. Isso elimina a necessidade de armazenar tokens no servidor, o que torna a autenticação mais escalável e fácil de gerenciar.

O Passport, por outro lado, é um framework de autenticação para aplicativos Node.js. Ele oferece uma ampla gama de estratégias de autenticação, incluindo autenticação baseada em senha, autenticação social (como login com o Facebook ou Google) e, é claro, autenticação com JWT. O Passport simplifica o processo de autenticação, fornecendo uma estrutura flexível e configurável que pode ser facilmente integrada em qualquer aplicativo Node.js.

Ao usar o Passport em conjunto com JWTs, é possível criar sistemas de autenticação altamente seguros e flexíveis. Os JWTs podem ser emitidos após um usuário fazer login com sucesso, e o Passport pode ser configurado para validar esses tokens em todas as rotas que requerem autenticação. Isso

permite que os desenvolvedores controlem facilmente o acesso a recursos protegidos, garantindo que apenas usuários autenticados e autorizados tenham acesso a eles.

Além disso, o uso de JWTs oferece a vantagem adicional de escalabilidade, pois os tokens são autocontidos e podem ser verificados sem a necessidade de consultar um banco de dados. Isso torna a autenticação mais eficiente em termos de desempenho, especialmente em sistemas com alto tráfego.

2.5 Gerenciamento de Dados e Banco de Dados

O gerenciamento de dados e o banco de dados desempenham um papel crítico em qualquer aplicativo ou sistema que envolva a coleta, armazenamento e recuperação de informações. No contexto de um servidor Node.js e usando o MySQL como banco de dados, o gerenciamento eficaz dos dados é essencial para a operação bem-sucedida do sistema. A integração do Prisma, simplifica significativamente essa tarefa.

O MySQL é um sistema de gerenciamento de banco de dados relacional amplamente utilizado, conhecido por sua confiabilidade, desempenho e escalabilidade. É especialmente adequado para aplicativos que exigem armazenamento estruturado de dados, como aplicativos de comércio eletrônico, sistemas de gerenciamento de conteúdo e muito mais. No contexto de um servidor Node.js, o MySQL pode ser acessado por meio de drivers e bibliotecas compatíveis, tornando a interação com o banco de dados eficiente e fácil de gerenciar.

O Prisma, por outro lado, é uma poderosa ORM que simplifica a comunicação entre o servidor Node.js e o MySQL. Ele oferece um mecanismo de consulta altamente intuitivo e baseado em tipos, que permite aos desenvolvedores criar, consultar e manipular dados de forma mais natural e eficaz. A principal vantagem do Prisma é a capacidade de gerar automaticamente consultas SQL complexas com base em consultas TypeScript altamente tipadas, garantindo a segurança e a integridade dos dados.

Ao utilizar o Prisma, os desenvolvedores podem definir modelos de dados TypeScript que mapeiam diretamente para tabelas no banco de dados MySQL. Isso elimina a necessidade de escrever manualmente consultas SQL complexas e torna a manutenção do código muito mais fácil. Além disso, o Prisma fornece uma camada de abstração que permite alternar facilmente entre diferentes bancos de dados, caso as necessidades do projeto mudem.

Em relação ao gerenciamento de dados, o MySQL oferece recursos robustos para garantir a integridade e a segurança dos dados. Isso inclui suporte para transações ACID (Atomicidade, Consistência, Isolamento e Durabilidade), autenticação avançada e recursos de controle de acesso, como permissões de usuário e criptografia de dados. Essas características são essenciais para proteger os dados do sistema contra ameaças e garantir a consistência das informações armazenadas.

No que diz respeito ao desempenho, o MySQL é altamente otimizado para consultas e operações de leitura e gravação eficientes. Ele suporta índices para acelerar a recuperação de dados e pode ser escalado horizontalmente para lidar com cargas de trabalho mais pesadas, se necessário.

Essas tecnologias são a base sólida para o desenvolvimento de aplicativos robustos e eficazes que podem atender às demandas do mundo real.

2.6 Escalabilidade

A escalabilidade é um fator crítico em qualquer sistema ou aplicativo que pretenda atender a um grande número de usuários, lidar com cargas de trabalho variáveis ou expandir-se com o tempo. No contexto de um servidor Node.js e do uso do MySQL como banco de dados, a escalabilidade é uma consideração fundamental para garantir que o sistema possa crescer de maneira eficaz e atender às demandas em constante mudança.

Existem duas formas principais de escalabilidade: escalabilidade vertical e escalabilidade horizontal. A escalabilidade vertical envolve aprimorar recursos em um único servidor, como aumentar a capacidade de processamento, a quantidade de memória ou a capacidade de armazenamento. No entanto, essa abordagem tem limitações físicas e pode se tornar cara à medida que os requisitos de recursos aumentam.

A escalabilidade horizontal, por outro lado, envolve a distribuição da carga de trabalho entre vários servidores. Isso é alcançado por meio de técnicas como balanceamento de carga, onde as solicitações dos usuários são distribuídas uniformemente entre vários servidores, e a adição de mais servidores conforme necessário para atender ao aumento da demanda.

No contexto do servidor Node.js, a escalabilidade horizontal é frequentemente a abordagem preferida, pois permite que o sistema cresça de maneira flexível e eficiente. O uso de bibliotecas e ferramentas, como o PM2, pode facilitar a escalabilidade horizontal, permitindo a execução de várias instâncias do Node.js em servidores diferentes e distribuindo automaticamente as solicitações dos usuários.

Quando se trata de escalabilidade do MySQL, várias estratégias podem ser implementadas para garantir que o banco de dados possa lidar com cargas de trabalho crescentes. Isso inclui o uso de técnicas como a replicação, onde cópias dos dados do banco de dados são mantidas em servidores secundários para leitura, aliviando a carga do servidor principal. Outra estratégia é o particionamento, onde os dados são divididos em fragmentos menores e distribuídos em servidores separados, permitindo uma distribuição equitativa da carga.

A implementação eficaz da escalabilidade envolve monitoramento constante do desempenho do sistema e a capacidade de dimensionar recursos conforme necessário. As ferramentas de monitoramento e análise de desempenho, juntamente com as práticas de desenvolvimento e configuração apropriadas, desempenham um papel fundamental na garantia de que o sistema seja capaz de crescer de forma eficiente e atender às demandas dos usuários.

3 Arquitetura do Servidor Node.js

As configurações do servidor foram projetadas com o objetivo de entregar uma arquitetura altamente organizada e estruturada, incorporando a linguagem TypeScript para o desenvolvimento de um sistema que prioriza a segurança, a tipagem estática, parâmetros bem definidos e funções com responsabilidades claramente definidas. Vamos explorar a arquitetura da aplicação e sua importância dentro do contexto geral do projeto.

3.1 Estrutura de Diretórios

- **Diretório de Rotas:** Este diretório desempenha um papel fundamental no roteamento e gerenciamento das requisições HTTP no servidor. Nele, são definidas as diferentes rotas às quais o aplicativo responderá, associando cada rota a um controlador ou ação específica. Isso contribui para manter o código organizado e separar as preocupações de forma eficaz.
- **Diretório de Modelos:** O diretório de modelos é onde ocorre a definição da estrutura dos dados a serem armazenados e manipulados pelo sistema. Os modelos representam os objetos com os quais o sistema lida, como entidades do banco de dados, e fornecem uma representação abstrata dos dados.
- **Diretório de Módulos:** Este é o diretório raiz que organiza todos os componentes específicos do sistema.
- **Diretório de Utilitários:** Neste diretório são disponibilizadas ferramentas úteis para o processamento dos dados e controle de acesso do sistema.

3.2 Classificação de Módulos

Dentro do diretório de módulos, segue-se uma classificação que se divide principalmente em duas categorias: Usuários e Pedidos. Essa abordagem de classificação é uma prática comum que torna o código mais legível e de fácil manutenção.

3.3 Diretórios de Repositórios

Dentro de cada categoria (Usuários e Pedidos), são criados diretórios adicionais para Repositórios. Essa prática é comum em arquiteturas de aplicativos modernos. Os repositórios têm a responsabilidade de isolar a lógica de acesso aos dados do restante do aplicativo. Dentro desses diretórios é possível encontrar:

- **Interfaces:** Elas definem os contratos que os repositórios devem seguir, facilitando a manutenção e a extensibilidade do sistema ao fornecer um conjunto claro de métodos que cada repositório deve implementar.

- **Repositórios:** Aqui, a lógica real para interagir com o banco de dados ou outra fonte de dados é escrita. Isso permite a isolação e a testagem separada dessa lógica em relação ao restante do sistema.

3.4 Diretórios de Casos de Uso e Controladores

Dentro de cada categoria (Usuários e Pedidos), também existem diretórios para Casos de Uso e Controladores, que são responsáveis pela lógica de negócios e pelo tratamento de solicitações HTTP, respectivamente. Isso segue uma abordagem de separação de preocupações, onde cada componente tem uma responsabilidade claramente definida:

- **Casos de Uso:** Neste local, a lógica de negócios da aplicação é implementada. Isso inclui validações, regras de negócios e interações com os repositórios. A separação dos casos de uso dos controladores permite a reutilização da lógica de negócios em diferentes contextos, como em uma API REST e em uma interface de linha de comando, por exemplo.
- **Controladores:** Os controladores são responsáveis por receber as solicitações HTTP, chamar os casos de uso apropriados e retornar as respostas adequadas. Eles lidam com a interação direta com o cliente, seja por meio de um navegador, um aplicativo móvel ou outro sistema, garantindo o tratamento correto das solicitações.
- **DTO (Data Transfer Objects):** DTOs são objetos usados para transferir dados entre os controladores e os casos de uso. Eles fornecem uma estrutura de dados independente do cliente e podem ser úteis para validar e normalizar os dados antes de serem processados pelos casos de uso.

4 Frontend com ReactJS

O ReactJS é uma das bibliotecas JavaScript mais populares e amplamente adotadas para o desenvolvimento de interfaces de usuário interativas e dinâmicas no lado do cliente. Criado e mantido pelo Facebook, o React, comumente chamado de ReactJS ou simplesmente React, revolucionou a forma como os desenvolvedores abordam o desenvolvimento de aplicações web modernas. Com seu paradigma de programação baseado em componentes, eficiência e reatividade, o React se tornou uma escolha dominante para construir interfaces de usuário de alta qualidade.

A abordagem principal do React é o conceito de componentização. Isso significa que os elementos da interface do usuário são divididos em pequenos componentes independentes, cada um com sua lógica, estrutura e funcionalidade específicas. Esses componentes podem ser compostos e reutilizados em toda a aplicação, o que leva a uma maior modularidade e facilita a manutenção do código.

Outro ponto forte do React é a eficiência na atualização da interface do usuário. O React adota uma técnica chamada de *Virtual DOM*, que permite atualizar apenas as partes da interface do usuário que realmente mudaram, em vez de redesenhar a página inteira. Isso resulta em uma experiência do usuário mais ágil e responsiva, mesmo em aplicações complexas.

O React também se destaca por sua comunidade ativa e pelo ecossistema rico de bibliotecas e ferramentas que o cercam. Há uma abundância de pacotes de terceiros, como o React Router para gerenciar as rotas da aplicação, o Redux para gerenciamento de estado, e muitos outros, que simplificam tarefas comuns no desenvolvimento de aplicações web.

Além disso, o React é altamente flexível e pode ser usado em diversas configurações. Ele pode ser integrado facilmente com outras tecnologias, como APIs REST, GraphQL e back-ends em Node.js, tornando-o uma escolha versátil para o desenvolvimento de front-ends de aplicativos web e móveis.

Uma das maiores vantagens do React é a sua capacidade de criar interfaces de usuário complexas de forma mais organizada e gerenciável. Os desenvolvedores podem dividir suas aplicações em componentes reutilizáveis, cada um focado em uma tarefa específica. Isso não apenas facilita o desenvolvimento, mas também torna a manutenção mais eficiente e escalável.

4.1 Visão Geral

Um front-end criado com ReactJS é uma parte essencial de muitas aplicações modernas, oferecendo uma experiência de usuário interativa e responsiva. Vamos dar uma visão geral de um front-end

ReactJS que utiliza as melhores práticas e tecnologias para fornecer uma experiência de usuário excepcional.

4.1.1 Linguagem de Programação: TypeScript

O front-end é desenvolvido em TypeScript, uma extensão do JavaScript que adiciona recursos de tipagem estática e ferramentas avançadas para desenvolvimento mais seguro e eficiente. O TypeScript ajuda a evitar erros comuns e oferece maior clareza no código.

4.1.2 Componentes e Biblioteca MUI

O front-end utiliza uma variedade de componentes para construir a interface do usuário de forma modular e reutilizável. A biblioteca Material-UI (MUI) é escolhida para fornecer componentes de alta qualidade, como botões, cartões, gráficos, formulários, cabeçalhos, inputs, barras laterais, caixas, grades e muito mais. Isso acelera o desenvolvimento e garante uma aparência consistente e atraente.

4.1.3 Hook de Autenticação

Para garantir a segurança e a autenticação dos usuários, o front-end utiliza um Hook de Autenticação. Isso permite que os usuários façam login de forma segura e acessem áreas restritas do aplicativo apenas quando estiverem autenticados.

4.1.4 Páginas e Rotas

O aplicativo possui várias páginas para atender às necessidades dos usuários. Isso inclui:

1. **Home:** Uma visão geral interativa que fornece informações essenciais do sistema.
2. **Calendário:** Uma ferramenta de calendário que ajuda na programação de eventos e tarefas.
3. **Link para GitHub:** Acesso ao GitHub do desenvolvedor do sistema.
4. **Usuários:** Gerenciamento de usuários, incluindo criação e edição de contas.
5. **Configurações:** Configurações gerais do aplicativo.
6. **Perfil de Usuário:** Página para que os usuários gerenciem suas informações pessoais.
7. **Criação de Usuários** Uma página para criar novos usuários no sistema.

Todas essas páginas são acessadas por meio de um sistema de rotas bem estruturado. As rotas garantem que os usuários sejam direcionados para a página correta com base em suas ações e permitem que a aplicação mantenha uma experiência de navegação suave e intuitiva.

4.1.5 Utilitários

Além disso, o front-end é aprimorado com utilitários que ajudam no desenvolvimento e na funcionalidade geral do aplicativo. Isso inclui funções de auxílio, ferramentas de formatação, funções de manipulação de dados e muito mais.

Em resumo, o front-end ReactJS descrito aqui representa uma abordagem robusta e eficaz para o desenvolvimento de interfaces de usuário modernas. Com TypeScript, componentes MUI de alta qualidade, autenticação segura, várias páginas, rotas bem definidas e utilitários úteis, ele oferece uma experiência de usuário rica e funcional para os usuários, enquanto simplifica o desenvolvimento e a manutenção para os desenvolvedores. Essa abordagem coloca a experiência do usuário e a qualidade do código em primeiro plano, garantindo que o aplicativo seja eficiente e confiável.

4.2 Componentes do Frontend

No desenvolvimento de um front-end ReactJS, a criação e a escolha dos componentes desempenham um papel fundamental na construção de uma interface de usuário eficaz e visualmente atraente. Uma abordagem comum é combinar componentes customizados com aqueles disponíveis em bibliotecas populares, como o Material-UI (MUI), para obter o melhor dos dois mundos.

4.2.1 Componentes Customizados:

Os componentes customizados são peças personalizadas de interface de usuário criadas especificamente para atender às necessidades do aplicativo. Eles podem ser projetados para se alinhar perfeitamente com a identidade visual da marca, fornecer funcionalidades específicas do domínio ou simplificar a interação do usuário com recursos exclusivos.

Ao criar componentes customizados, os desenvolvedores têm total controle sobre o design e o comportamento, garantindo que atendam aos requisitos específicos do projeto. Esses componentes podem ser facilmente reutilizados em várias partes do aplicativo, promovendo a consistência e economizando tempo de desenvolvimento.

4.2.2 Componentes do Material-UI (MUI):

O Material-UI é uma biblioteca popular de componentes de interface de usuário que segue as diretrizes de design do Material Design, do Google. Ela oferece uma ampla gama de componentes prontos para uso, como botões, cards, gráficos, formulários, barras laterais, grids e muito mais. Esses componentes são altamente personalizáveis e vêm com estilos responsivos integrados, tornando-os ideais para criar uma interface de usuário moderna e atraente.

A principal vantagem de utilizar componentes de bibliotecas como o MUI é a economia de tempo. Em vez de criar componentes personalizados para cada parte da interface de usuário, os desenvolvedores podem aproveitar os componentes já existentes e personalizá-los conforme necessário. Isso acelera o desenvolvimento e garante uma aparência consistente em todo o aplicativo.

Além disso, bibliotecas como o MUI são mantidas ativamente pela comunidade de desenvolvedores e recebem atualizações regulares de segurança e melhorias de desempenho. Isso significa que os desenvolvedores podem contar com componentes confiáveis e manter seus aplicativos atualizados com as últimas melhorias sem muito esforço.

4.3 Tour do site

A utilização de ferramentas de tour de site, como a biblioteca intro.js, desempenha um papel significativo na melhoria da experiência do usuário e na eficácia do aplicativo. Esse tipo de recurso é especialmente valioso em aplicações complexas, onde a compreensão dos recursos disponíveis e a navegação adequada são essenciais.

4.3.1 Compreensão dos Botões na Barra Lateral

A barra lateral de um sistema de gerenciamento geralmente contém uma série de botões e opções que permitem aos usuários navegar pelo sistema, acessar diferentes funcionalidades e visualizar informações específicas. Essa barra lateral pode ser um elemento crítico da interface do usuário, mas, às vezes, a complexidade da interface pode ser um desafio para os usuários.

É aí que o tour do site se torna valioso. Ao criar um tour que destaca cada botão e fornece informações contextuais sobre sua função, os usuários podem rapidamente entender como usar a barra lateral e explorar as opções disponíveis. O intro.js permite criar dicas interativas que orientam os usuários passo a passo, destacando os botões e explicando suas finalidades. Isso torna a barra lateral mais acessível, reduz a curva de aprendizado e ajuda os usuários a navegar de maneira mais eficaz.

4.3.2 Visualização de Itens em Cartões e Imagens:

Novamente, o tour do site oferece uma solução eficaz. À medida que os usuários exploram o dashboard, o intro.js pode destacar cada cartão ou gráfico individualmente, fornecendo explicações detalhadas sobre o que os dados representam e como interpretá-los. Isso não apenas ajuda os usuários a compreender as informações apresentadas, mas também os capacita a tomar decisões informadas com base nos dados.

5 Notas de Desenvolvimento do Backend

O Prisma é uma ferramenta ORM (Object-Relational Mapping) que facilita a interação com bancos de dados em projetos Node.js. Para configurar o Prisma com SQLite vamos seguir os passos abaixo:

5.1 Configuração Inicial

Partimos do pressuposto que o Node.js já esteja instalado no sistema, pois, o projeto pode ser executado tanto do Windows, do Linux quanto do MacOS. Certifique-se de que você já possui o Node.js instalado em seu sistema. Se não, você pode baixá-lo em https://nodejs.org/.

5.2 Inicialização do Projeto

Inicialize um novo projeto Node.js no diretório do seu projeto, se ainda não tiver feito isso. Use o seguinte comando:

```
    npm init -y
```

6 Instalação do Prisma e SQLite

Instale o Prisma, o SQLite e outras dependências necessárias usando o npm:

```
   npm install @prisma/cli @prisma/client sqlite3
```

6.1 Configuração do Prisma

Execute o seguinte comando para configurar o Prisma no seu projeto:

```
    npx prisma init
```

Isso criará um arquivo ‘schema.prisma‘ no seu projeto. Abra este arquivo e configure o banco de dados SQLite, por exemplo:

```
// schema.prisma

generator client {
  provider = "prisma-client-js"
  output   = "./generated/client"
}

// Defina a conexão com o banco de dados SQLite
datasource db {
  provider = "sqlite"
  url      = "file:./dev.db" // O nome do arquivo do banco de dados
     SQLite
}

// Defina seu modelo de prato (exemplo)
model Dish {
  id          Int      @id @default(autoincrement())
  name        String
  category    String
  description String
  ingredients String[]
  price       Float
  imageUrl    String
}
```

6.2 Gerando o Cliente Prisma

Execute o seguinte comando para gerar o cliente Prisma com base no seu modelo:

```
    npx prisma generate
```

6.3 Uso do Cliente Prisma

Agora, você pode usar o cliente Prisma gerado para interagir com o banco de dados SQLite em seu código Node.js. Por exemplo, para criar um novo prato no banco de dados:

```
const { PrismaClient } = require('@prisma/client');
const prisma = new PrismaClient();

async function createDish() {
  const newDish = await prisma.dish.create({
    data: {
      name: 'Nome do Prato',
      category: 'Categoria',
      description: 'Descrição do Prato',
      ingredients: ['Ingrediente 1', 'Ingrediente 2'],
      price: 19.99,
      imageUrl: 'URL da Imagem',
    },
  });

  console.log('Prato criado:', newDish);
}

createDish()
  .catch((error) => {
    throw error;
  })
  .finally(async () => {
    await prisma.$disconnect();
  });
```

Lembre-se de adaptar o código acima para suas necessidades específicas, criando funções para listar, editar ou excluir pratos, conforme necessário.

Com essas etapas, você deve estar pronto para começar a usar o Prisma com SQLite em seu projeto "food explorer". Certifique-se de consultar a documentação oficial do Prisma para obter mais detalhes sobre seu uso: https://www.prisma.io/docs/

7 Notas de Desenvolvimento do Frontend

Para criar o frontend do "food explorer"usando React.js, você pode seguir os seguintes passos:

Passo 1: Inicialização do Projeto

Certifique-se de ter o Node.js instalado em seu sistema. Em seguida, você pode criar um novo projeto React com o seguinte comando:

"'bash npx create-react-app food-explorer-frontend "'

Substitua "food-explorer-frontend"pelo nome que desejar para o seu projeto. Isso criará um novo diretório com a estrutura inicial do projeto React.

Passo 2: Navegação para o Diretório do Projeto

Navegue para o diretório recém-criado do seu projeto:

"'bash cd food-explorer-frontend "'

Passo 3: Estrutura de Pastas e Arquivos

Dentro do diretório do seu projeto React, você encontrará uma estrutura de pastas e arquivos semelhante a esta:

"' food-explorer-frontend/ node$_modules/public/index.htmlfavicon.ico...src/index.jsApp.jscomponents/....gitignore$

A pasta 'public' contém o arquivo 'index.html', que é o ponto de entrada da sua aplicação React. A pasta 'src' é onde você colocará seus componentes React e lógica de frontend.

Passo 4: Desenvolvimento Local

Inicie o servidor de desenvolvimento local com o seguinte comando:

“‘bash npm start “‘

Isso iniciará o servidor e abrirá automaticamente sua aplicação no navegador.

Passo 5: Desenvolvimento

Agora, você pode começar a desenvolver o frontend da aplicação ”food explorer”usando React.js. Você pode criar componentes React em ‘src/components‘, definir rotas, estilizar usando CSS ou pré-processadores como SASS e conectar-se ao backend (API Prisma) conforme necessário.

Lembre-se de consultar a documentação do React para obter mais informações sobre como criar componentes, definir rotas e gerenciar estado:

- Documentação do React: https://reactjs.org/docs/getting-started.html

Esses são os passos básicos para criar um projeto React.js do zero. À medida que você avança no desenvolvimento, adapte a estrutura de pastas e arquivos de acordo com as necessidades específicas do seu projeto ”food explorer”. Boa sorte com o desenvolvimento do frontend em React!

8 Conclusão

Concluindo, esta documentação abrangeu aspectos cruciais tanto do backend quanto do frontend de nosso sistema de Gerenciamento de Restaurantes. Ao longo deste documento, exploramos a arquitetura do servidor Node.js, a organização de diretórios, a importância da segurança e autenticação com JWT e Passport, bem como a gestão de dados por meio do MySQL e do Prisma. Além disso, discutimos a escalabilidade do sistema, detalhando como as configurações podem ser ajustadas para atender às crescentes demandas.

No frontend, destacamos a utilização do ReactJS, incluindo a estruturação do projeto com componentes personalizados e componentes disponíveis no Material-UI (MUI). Também exploramos como o TypeScript é uma ferramenta valiosa para manter um código seguro e tipado. Além disso, apresentamos a funcionalidade de tour do site utilizando a biblioteca intro.js, que desempenha um papel crucial na orientação dos usuários e no aprimoramento da experiência geral.

Esta documentação visa fornecer um guia abrangente para desenvolvedores, administradores e outros membros da equipe envolvidos na manutenção e expansão do sistema. Ao seguir as melhores práticas apresentadas aqui, esperamos que nossa plataforma de gerenciamento seja eficiente, segura e escalável, atendendo às necessidades de nossos usuários e contribuindo para um impacto positivo no combate às mudanças climáticas. Continuaremos a aprimorar e atualizar esta documentação à medida que nosso sistema evolui, garantindo que ela permaneça uma referência valiosa para todos os envolvidos em nosso projeto.

9 Referências

1. Documentação do Node.js: https://nodejs.org/
2. documentação do Express: https://expressjs.com/pt-br/
3. documentação do PassportJS: https://www.passportjs.org/
4. Documentação do Prisma: https://www.prisma.io/docs/
5. Documentação do ReactJS: https://reactjs.org/docs/getting-started.html
6. Documentação do MUI: https://mui.com/
7. Documentação do IntroJS: https://introjs.com/