Relatório Técnico: Aplicação Web de Clima

1. Introdução

Este relatório detalha as decisões técnicas, a arquitetura e os desafios encontrados durante o desenvolvimento de uma aplicação web simples de clima. O objetivo principal foi criar uma solução com frontend e backend separados, capaz de consumir uma API externa (OpenWeatherMap) para fornecer informações meteorológicas.

2. Tecnologias Utilizadas

A escolha das tecnologias foi baseada na popularidade, na capacidade de desenvolvimento rápido e na separação clara de responsabilidades entre o frontend e o backend.

2.1. Backend

- **Node.js:** Escolhido como ambiente de execução para o backend devido à sua natureza assíncrona e orientada a eventos, o que o torna eficiente para operações de I/O intensivas, como chamadas a APIs externas. A utilização de JavaScript em todo o stack (frontend e backend) também simplifica o contexto de desenvolvimento.
- **Express.js:** Um framework web minimalista e flexível para Node.js. Foi utilizado para criar a API RESTful, facilitando a definição de rotas, o tratamento de requisições HTTP e o envio de respostas JSON.
- **CORS (Cross-Origin Resource Sharing):** Essencial para permitir que o frontend (rodando em uma porta diferente) faça requisições ao backend sem ser bloqueado pelas políticas de segurança do navegador. O middleware cors foi empregado para configurar isso de forma simples.
- **node-fetch (ou fetch nativo do Node.js):** Utilizado para fazer as requisições HTTP da API OpenWeatherMap a partir do backend. Em versões mais recentes do Node.js, o fetch é nativo, simplificando ainda mais.

2.2. Frontend

- **React:** Um framework JavaScript para construção de interfaces de usuário. Foi escolhido por sua abordagem baseada em componentes, reatividade e eficiência na atualização do DOM. A utilização de componentes funcionais e Hooks (useState) permitiu um gerenciamento de estado simples e direto.
- **HTML/CSS Puro:** Para a estilização da interface, optou-se por CSS puro

2.3. API Externa

- **OpenWeatherMap API:** A fonte dos dados meteorológicos. A API fornece endpoints para clima atual, previsão do tempo e geocodificação (para obter

coordenadas de cidades). A chave da API foi fornecida e utilizada nas requisições do backend.

3. Estrutura da Aplicação

A aplicação segue uma arquitetura cliente-servidor com uma clara separação de preocupações.

3.1. Backend (server.js)

O arquivo server.js atua como o coração do backend.

- **Inicialização:** Configura o servidor Express e o middleware CORS.
- **Endpoints RESTful:** Foram implementados três endpoints principais:
    - GET /api/weather/current?city={nome_da_cidade}: Retorna os dados do clima atual para a cidade especificada.
    - GET /api/weather/forecast?city={nome_da_cidade}: Retorna a previsão do tempo para os próximos 5 dias (com intervalos de 3 horas) para a cidade especificada.
    - GET /api/weather/city-coords?city={nome_da_cidade}: Retorna as coordenadas geográficas (latitude e longitude) da cidade especificada.
- **Consumo da API Externa:** Cada endpoint do backend faz uma requisição fetch para a API OpenWeatherMap, repassando a chave da API e os parâmetros necessários.
- **Tratamento de Respostas:** As respostas da OpenWeatherMap são processadas e retransmitidas para o frontend, incluindo tratamento básico de erros (por exemplo, cidade não encontrada).

3.2. Frontend (App.js)

O arquivo App.js é o componente principal da aplicação React.

- **Gerenciamento de Estado:** Utiliza o Hook useState para gerenciar o nome da cidade inserido pelo usuário, os dados do clima atual, a previsão do tempo, as coordenadas da cidade, o estado de carregamento e mensagens de erro.
- **Interface do Usuário:** Contém um campo de entrada para o nome da cidade e três botões para acionar as diferentes requisições à API (Clima Atual, Previsão e Coordenadas).
- **Comunicação com o Backend:** As funções fetchCurrentWeather, fetchForecast e fetchCityCoords são responsáveis por fazer requisições HTTP para os endpoints do backend.

- **Renderização Condicional:** Os dados recebidos do backend são exibidos na interface de forma condicional, mostrando mensagens de carregamento ou erro conforme necessário.
- **Estilização:** Os estilos CSS são aplicados diretamente no arquivo App.css, garantindo que a interface seja visualmente agradável e responsiva.

4. Dificuldades Enfrentadas e Soluções

Durante o desenvolvimento, algumas dificuldades comuns em aplicações web foram encontradas e resolvidas:

4.1. Problemas de CORS (Cross-Origin Resource Sharing)

- **Dificuldade:** Ao tentar fazer requisições do frontend (rodando em http://localhost:3000) para o backend (rodando em http://localhost:3001), o navegador impõe restrições de segurança que impedem requisições entre origens diferentes. Isso resultaria em erros de CORS.
- **Solução:** A instalação e configuração do middleware cors no servidor Express (app.use(cors());) resolveu o problema, permitindo que o backend aceite requisições do frontend.

4.2. Gerenciamento de Requisições Assíncronas

- **Dificuldade:** Lidar com chamadas de API assíncronas no JavaScript pode levar a código complexo com callbacks aninhados (callback hell).
- **Solução:** A utilização de async/await tanto no backend (para chamar a OpenWeatherMap) quanto no frontend (para chamar o backend) tornou o código mais legível e fácil de manter, permitindo que as operações assíncronas sejam escritas de forma síncrona.

4.3. Tratamento de Erros da API Externa

- **Dificuldade:** As APIs externas podem retornar erros (por exemplo, cidade não encontrada, chave de API inválida). É crucial tratar esses erros para fornecer feedback útil ao usuário.
- **Solução:** Blocos try-catch foram implementados em todas as chamadas fetch no backend e no frontend. As respostas da API OpenWeatherMap são verificadas (response.ok), e mensagens de erro são passadas para o frontend, que as exibe ao usuário.

5. Conclusão

A aplicação web de clima foi desenvolvida com sucesso, demonstrando a integração de um frontend React com um backend Node.js para consumir uma API externa. As decisões técnicas foram tomadas visando a clareza do código, a modularidade e a capacidade de expansão futura. Os desafios comuns de desenvolvimento web,

como CORS e gerenciamento assíncrono, foram abordados com soluções padrão da indústria, enquanto problemas específicos do ambiente de compilação foram contornados com adaptações pragmáticas.