ADR: Todo-List

Integrantes: Luiz Disarz e Matheus Finatto

Contexto

Estamos desenvolvendo uma aplicação web de lista de tarefas que permite que os usuários gerenciem e acompanhem suas tarefas. Precisamos escolher uma pilha front-end que permita um desenvolvimento eficiente, forneça uma interface de usuário robusta e responsiva e permita integração com nossa API back-end. Já para o back-end, precisamos escolher tercnologias confiáveis, velozes e robustas para maximizar os resultados que buscamos. Avaliamos diferentes opções e consideramos fatores como facilidade de uso, experiência do desenvolvedor, suporte da comunidade e compatibilidade do ecossistema.

Baseando-se nisso, decidimos usar a seguinte pilha front-end para nossa aplicação de lista de tarefas:

- Vite: Como ferramenta de compilação, fornecendo tempos rápidos de desenvolvimento e compilação com seu empacotamento nativo baseado em módulos ES.
- React: Como biblioteca JavaScript para construir a interface do usuário, oferecendo uma abordagem baseada em componentes e um ecossistema vibrante.
- TypeScript: Para tipagem estática e melhor experiência de desenvolvedor, melhorando a qualidade do código e fornecendo um melhor suporte de ferramentas.
- Fetch API: Para realizar solicitações HTTP à API back-end, oferecendo uma forma moderna e padronizada de realizar operações de rede.
- Vitest: Uma biblioteca de testes de unidade extremamente rápida e poderosa, que aproveita a infraestrutura e as vantagens do Vite. Ela fornece uma API compatível com o Jest, incluindo recursos comuns como mocking, snapshots e cobertura de código.

Para a camada back end, decidimos adicionar o seguinte:

- Spring Boot: Como framework para simplificar o desenvolvimento da API em Java, oferecendo recursos como roteamento, gerenciamento de dependências e integração com o Spring Data.
- PostgreSQL: Como banco de dados relacional para persistir e recuperar os dados das tarefas.

Consequências

A pilha front-end escolhida oferece diversos benefícios e motivos para sua escolha: - Os tempos rápidos de compilação e a substituição de módulo em tempo real do Vite aumentarão a produtividade do desenvolvedor e permitirão uma experiência de desenvolvimento tranquila. - A arquitetura baseada em compo-

nentes do React permitirá a reutilização de código, facilitando a manutenção e a expansão da aplicação. - A tipagem estática do TypeScript ajudará a identificar erros durante o desenvolvimento, melhorará a qualidade do código e fornecerá um melhor suporte de ferramentas. - A Fetch API fornece uma forma moderna e padronizada de realizar solicitações HTTP, permitindo uma comunicação perfeita com a API back-end. - Familiaridade com as tecnologias e importância de conhece-lás no mercado profissional.

Ao adicionar o Spring Boot e o PostgreSQL, obteremos os seguintes benefícios para a camada de backend: - O Spring Boot simplificará o desenvolvimento da API em Java, fornecendo recursos poderosos e uma estrutura bem definida. - O PostgreSQL é um banco de dados relacional confiável e robusto, oferecendo recursos avançados de gerenciamento de dados.

Alternativas

As pilhas de tecnologias alternativas consideradas foram, para front-end: - Angular: Embora o Angular seja um framework abrangente com um ecossistema forte, optamos pelo React devido à sua simplicidade e facilidade de uso para este projeto. - Vue.js: O Vue.js é outro framework JavaScript popular que oferece funcionalidades semelhantes ao React. No entanto, escolhemos o React devido à sua comunidade maior e ecossistema mais rico, que está mais alinhado com nossos requisitos de projeto. - Jest: Uma biblioteca de testes de JavaScript amplamente adotada pela comunidade. Embora tenha sido considerada como uma opção alternativa, decidimos usar o Vitest devido à sua integração nativa com o Vite e ao seu desempenho superior, além de ser uma tecnologia nova.

E para back-end: - Express.js com MongoDB: Uma combinação popular para o desenvolvimento de APIs em Node.js, com o MongoDB como banco de dados NoSQL. Essa alternativa oferece flexibilidade e escalabilidade, especialmente para aplicativos com requisitos de dados não estruturados. Ficou em segundo lugar na lista de tecnlogias, mas no final, foi optado por seguir com Spring + Postgres por questões como possível necessidade de auxilio advindo do professor. - Django com SQLite: O Django é um framework Python robusto para desenvolvimento web, e o SQLite é uma opção leve de banco de dados relacional. Essa combinação é fácil de usar e adequada para projetos menores ou de escopo limitado. Não foi escolhido pela falta de familiaridade com a ferramenta, e pelo projeto integrador ter sido feito com Flask, um framework de Python e SQLite.

Como executar

- Banco de Dados: É necessário criar uma database chamada de `todos` em postgres. Caso sejam necessárias, as credências e demais específicações estão dentro do arquivo application.yaml (back/src/main/resources/application.yaml);
- Backend: É necessário o Java, e recomendado o uso de uma IDE especializada, como IntelliJ. Com ela, basta executar o arquivo Application.java (back/src/main/java/br/com/uri/spring/Application.java) (ficará ex-

posto em http:/localhost:8080/v1). Para rodar os testes unitários do backend, basta executar os testes a partir da pasta java dentro de test (back/src/test/java);
- Frontend: Para executar os testes unitários basta executar o comando `npm test` e para executar o servidor frontent o comando `npm run dev` é necessário (ficará exposto em http:/localhost:5173).

Referências

- Documentação do Vite: https://vitejs.dev/
- Documentação do React: https://reactjs.org/
- Documentação do TypeScript: https://www.typescriptlang.org/
- Documentação do Fetch API: https://developer.mozilla.org/pt-BR/docs/Web/API/Fetch_API
- Documentação do Vitest: https://vitejs.dev/
- Documentação do Spring Boot: https://spring.io/projects/spring-boot
- Documentação do PostgreSQL: https://www.postgresql.org/

Para ver o repositório no Github clique aqui

Ou acesse este link: https://github.com/MatheusFinatto/prog-web-trabalho-final