Especificação da Arquitetura Frontend

1. Princípios e Estratégia

O frontend do AxonAI será desenvolvido seguindo quatro princípios fundamentais:

1. **Arquitetura Baseada em Componentes:** A UI será decomposta em componentes reutilizáveis, com responsabilidades bem definidas, facilitando a manutenção e a consistência visual.
2. **Segurança de Tipos (*Type Safety*):** O uso de TypeScript é mandatório para garantir a robustez, reduzir bugs em tempo de execução e melhorar a experiência de desenvolvimento através da análise estática.
3. **Estado Reativo e Previsível:** A gestão de estado será claramente dividida entre estado do servidor (dados da API) e estado do cliente (UI), utilizando bibliotecas especializadas para cada um.
4. **Separação de Responsabilidades:** A lógica de UI (componentes), a lógica de negócio do cliente (hooks) e as chamadas de serviço (API) serão mantidas em ficheiros e pastas separadas.

2. Tecnologias e Bibliotecas

A stack tecnológica foi escolhida para maximizar a produtividade do desenvolvedor e a robustez da aplicação.

- **Framework:** React 19 com Vite
- **Linguagem:** TypeScript
- **Gestão de Estado do Servidor:** TanStack Query (React Query)
- **Gestão de Estado Global do Cliente:** Zustand
- **Biblioteca de Componentes:** Shadcn/ui
- **Chamadas HTTP:** Axios
- **Roteamento:** React Router

3. Estratégia de Gestão de Estado

Para garantir a previsibilidade e evitar complexidade, a gestão de estado seguirá uma hierarquia clara:

1. **Estado do Servidor (TanStack Query):** É a fonte da verdade para **qualquer dado que venha da API**. TanStack Query será responsável por fetching, caching, sincronização e atualização de dados do servidor. Deve-se evitar duplicar este estado em qualquer outro gestor de estado.
2. **Estado Global do Cliente (Zustand):** Para estado que **não vem do servidor**, mas precisa ser partilhado por toda a aplicação. Exemplos: estado de autenticação do utilizador (ex: `isAuthenticated`, `userProfile`), tema da UI (claro/escuro), estado de um tutorial interativo.
3. **Estado Local (Component State - `useState` / `useReducer`):** Para estado que é confinado a um único componente ou a uma pequena árvore de componentes que podem partilhá-lo via *props*. Exemplos: estado de um formulário antes de ser submetido, se um modal está aberto ou fechado, o valor de um campo de input.

4. Estrutura de Pastas (Mantida)

A estrutura de pastas organizada é fundamental para a escalabilidade.

```
/src
/api            # Módulos de chamada à API, organizados por recurso
/components     # Componentes de UI reutilizáveis (Shadcn/ui e customizados)
/features       # Módulos de funcionalidades (ex: autenticação, projeto)
/hooks          # Hooks customizados e reutilizáveis
/lib            # Configurações de bibliotecas (axios.ts, etc.)
/pages          # Componentes que representam as páginas da aplicação
/providers      # Provedores de contexto da aplicação (React Query, Router)
/store          # Lojas do Zustand para estado global
/types          # Definições de tipos TypeScript, geradas a partir da API
```

5. Sincronização de Tipos com a API (Type Generation)

Para garantir a consistência de tipos entre o frontend e o backend, os tipos do TypeScript serão gerados automaticamente a partir da especificação OpenAPI 3 do backend.

- **Ferramenta:** `openapi-typescript` ou similar.
- **Fluxo de Trabalho:**
    1. O backend expõe a especificação `openapi.json` através do `springdoc-openapi`.

2. Um script no `package.json` do frontend (`npm run generate-api-types`) irá buscar esta especificação e gerar um ficheiro `api-types.ts` no diretório `/src/types`.
3. Este script será executado durante o processo de build e pode ser executado manualmente sempre que o contrato da API for alterado, garantindo que o frontend esteja sempre sincronizado com a API.

6. Estratégia de Testes

A aplicação terá uma cobertura de testes robusta para garantir a fiabilidade e facilitar a refatoração.

- **Ferramentas:**
  - **Test Runner:** Vitest.
  - **Testes de Componentes:** React Testing Library.
  - **Mocking de API:** Mock Service Worker (MSW).
  - **Testes End-to-End (E2E):** Playwright.
- **Níveis de Teste:**
  1. **Testes de Integração (Foco Principal):** Testaremos componentes ou páginas inteiras renderizadas no DOM virtual. As interações do utilizador serão simuladas e as chamadas à API serão interceptadas e "mockadas" com o MSW. O objetivo é testar o comportamento da aplicação da perspetiva do utilizador, sem depender de um backend real.
  2. **Testes Unitários:** Para lógica de negócio complexa isolada em hooks ou funções utilitárias puras.
  3. **Testes End-to-End (E2E):** Um conjunto pequeno e crítico de testes que verificam os fluxos de trabalho mais importantes (ex: login completo, criação de um projeto, proposta de tarefas pela IA) contra um ambiente de *staging* com um backend real.

7. Qualidade de Código e Convenções

Ferramentas automatizadas serão usadas para manter a qualidade e a consistência do código.

- **Linting:** ESLint com plugins para React, TypeScript e acessibilidade (a11y) para detetar problemas de código estaticamente.

- **Formatação:** Prettier para garantir um estilo de código consistente em toda a base de código.
- **Git Hooks:** Husky e `lint-staged` serão configurados para executar o ESLint e o Prettier automaticamente antes de cada `commit`, prevenindo que código com problemas de qualidade seja enviado para o repositório.
- **Convenções de Commit:** A equipa adotará o padrão **Conventional Commits** para as mensagens de `commit` (`feat:`, `fix:`, `chore:`, etc.). Isto permite a geração automática de `CHANGELOGs` e facilita a compreensão do histórico do projeto.

Ensino e Análise Final

Uma arquitetura de frontend moderna e escalável é um ecossistema, não apenas uma biblioteca. Escolher React é o primeiro passo, mas o que garante o sucesso do projeto a longo prazo são as **regras e ferramentas** que governam como o código é escrito, testado e mantido. Ao definir estratégias claras para gestão de estado, testes automatizados e qualidade de código, criamos "guardrails" que permitem aos desenvolvedores construir novas funcionalidades com velocidade e confiança. A automação da geração de tipos a partir da especificação OpenAPI é a ponte que solidifica a colaboração entre frontend e backend, eliminando uma classe inteira de bugs de integração e garantindo que ambas as equipas falem a mesma "linguagem".