Documento de Arquitetura de Software

Plataforma FitTrack

Professor: João Paulo Coelho Furtado

Alunos: Cecília Fernandes, Isabela Demaria, João Victor Martins e Laura Portugal

1. Introdução

1.1. Objetivo

O objetivo do sistema FitTrack é oferecer uma plataforma digital intuitiva e eficiente para o monitoramento de atividades físicas, permitindo que os usuários acompanhem sua evolução ao longo do tempo. A solução visa promover hábitos saudáveis e incentivar a consistência na prática de atividades físicas, oferecendo recursos motivacionais e uma experiência de uso acessível e segura.

1.2. Escopo

O sistema FitTrack será desenvolvido como uma aplicação web responsiva voltada para o monitoramento de atividades físicas, com funcionalidades de cadastro de usuários, registro manual de atividades, definição de metas recorrentes e visualização de progresso. O público-alvo inclui usuários finais, equipe de desenvolvimento, stakeholders e clientes.

2. Visão geral da Arquitetura

2.1. Estilo Arquitetural

O sistema será desenvolvido seguindo o estilo arquitetural em camadas (Layered Architecture), visando separação de responsabilidades, manutenibilidade e simplicidade de implementação. A arquitetura adota o modelo MVC simplificado, dividindo-se em:

- Apresentação (Front-End): interface do usuário para acesso via web responsiva.
- Negócios (Back-End): regras de negócio, controle de fluxo e serviços principais.
- Persistência (Banco de Dados): armazenamento de dados estruturados em SGBD relacional.

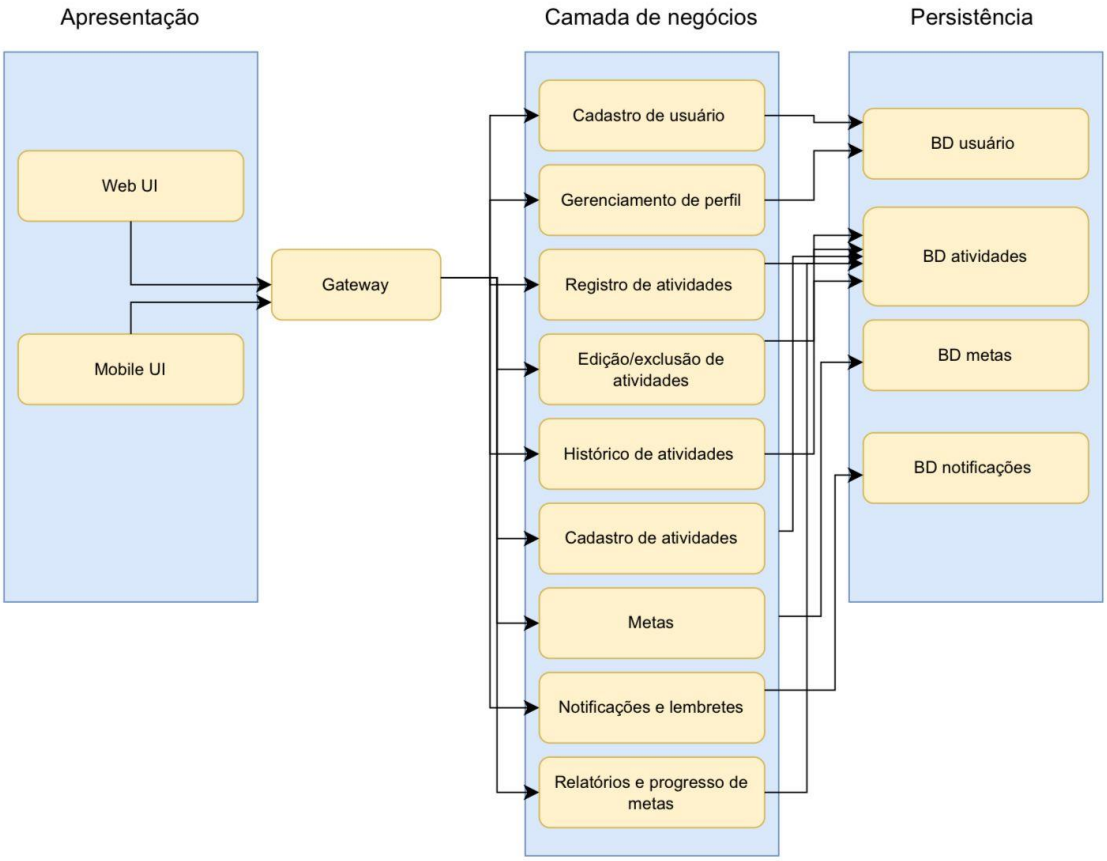
2.2. Camadas

- Camada de Apresentação: responsável pela interação com o usuário (Web).
- Camada de Negócios: implementa a lógica do sistema (cadastro, registro de atividades, metas e notificações).
- Camada de Persistência: responsável pela comunicação com o banco de dados.

3. Componentes Principais

ID	Componente	Descrição
C001	UsuarioController	Expõe endpoints REST para cadastro, login, edição de perfil e recuperação de senha.
C002	AtividadeController	Gerencia endpoints para registro, edição, exclusão e histórico de atividades.
C003	MetaController	Expõe endpoints para criação, edição, exclusão e acompanhamento de metas recorrentes.
C004	NotificacaoController	Forendpoints para configuração e envio de notificações/lembretes.
C005	UsuarioService	Contém regras de negócio relacionadas ao usuário, incluindo autenticação e validação de dados.
C006	AtividadeService	Implementa a lógica de negócios para cadastro, edição e análise de atividades físicas.
C007	MetaService	Contém regras para definição, renovação e progresso de metas recorrentes.
C008	NotificacaoService	Regras de envio de notificações automáticas e personalizadas.
C009	UsuarioRepository	Repositório responsável pela persistência de dados do usuário.
C0010	AtividadeRepository	Repositório responsável pela persistência de registros de atividades.
C0011	MetaRepository	Repositório para persistência e consulta de metas.
C0012	NotificacaoRepository	Repositório para armazenamento de notificações e histórico de envios.
C0013	RelatorioService	Gera dados consolidados de progresso e estatísticas de atividades/metas.

3.1. Diagrama de Componentes



4. Tecnologias Utilizadas

ID	Camada	Tecnologias
001	Usabilidade	O sistema deve ter interface simples e responsiva.
002	Front-End	Dart/Flutter ou React.js

003	Back-End	Node.js
004	Banco de Dados	PostgreeSql
005	Versionamento	Git, GitHub
006	Desenvolvimento	Visual Studio, VSCode
007	Autenticação	OAuth 2.0 / OpenID Connect

5. Padrões e Convenções

- Boas práticas de código:
  - SOLID: aplicar princípios como *Single Responsibility* e *Dependency Inversion* para manter o código desacoplado e mais fácil de testar.
  - Clean Code: nomes claros, funções pequenas, evitar duplicação de lógica.
  - Clean Architecture: separar bem as camadas para não misturar regras de negócio com detalhes técnicos.
  - Controle de dependências: usar apenas bibliotecas realmente necessárias e mantê-las atualizadas.
- Boas práticas:
  - Linters (ESLint, dart analyze) para impor regras de sintaxe e estilo.
  - Guia de estilo interno: documentar convenções de nome de variáveis, organização de pastas, estrutura de commits.
  - Code Review – adotar revisão obrigatória antes do merge.
- Uso de DTO (Data Transfer Objects):
  - Separar DTO de Entidade: nunca expor diretamente suas entidades de banco ou de domínio em requisições/respostas.
  - Validação: usar DTOs para validar entrada e saída de dados (package:freezed no Dart).
  - Segurança: ao expor dados, enviar só os campos necessários (ex.: nunca retornar senha no DTO de usuário).
- Extras:
  - Monitoramento & Logging: padronizar logs (JSON, níveis de log), usar tracing com OpenTelemetry.
  - Feature flags: ativar/desativar recursos sem precisar refazer Deploy.
  - CI/CD: configurar pipeline com lint, teste e build para garantir que só código saudável vai para produção.

6. Requisitos Não Funcionais

ID	Requisitos Categoria ISO 9126	Descrição
RNF001	Usabilidade	O sistema deve ter interface simples e responsiva.
RNF002	Segurança	As senhas devem ser armazenadas de forma criptografada.
RNF003	Adaptabilidade	O sistema deve ser responsivo (funcionar bem em computadores e celulares).
RNF004	Eficiência	O sistema deve registrar atividades em menos de 2 segundos.
RNF005	Compatibilidade	O sistema deve ser compatível com navegadores modernos (Chrome, Edge, Firefox).
RNF006	Segurança/Privacidade	Os dados do usuário devem ser armazenados e tratados conforme a LGPD.
RNF007	Escalabilidade	O Sistema deve suportar 10.000 usuários simultâneos, sem degradação de desempenho
RNF008	Manutenibilidade	O Sistema deve ser construído seguindo padrões de código que facilitem sua manutenção e evolução.

7. Riscos Arquiteturais

- Complexidade na lógica de metas recorrentes: O reinício automático e o acompanhamento por período exigem regras bem definidas e testes rigorosos.
- Escalabilidade do banco de dados: O crescimento do número de usuários e registros pode exigir otimizações futuras na estrutura de dados e nas consultas.
- Integração com serviços externos: Notificações, autenticação via OAuth/OpenID e envio de e-mails dependem de serviços de terceiros, sujeitos a instabilidade ou mudanças de API.

- Conformidade com LGPD: O tratamento de dados pessoais e de saúde exige atenção especial à privacidade, consentimento e segurança.
- Responsividade e desempenho em dispositivos móveis: A experiência do usuário pode ser impactada se a interface não for bem otimizada para diferentes resoluções e navegadores.

8. Decisões Arquiteturais

ID	Decisão	Justificativa
001	Uso de Node.js com TypeScript no backend	Proporciona alta performance, tipagem segura, ampla comunidade e suporte a APIs REST.
002	Escolha de React.js ou Flutter no frontend	Permite interfaces responsivas e modernas, com boa experiência em dispositivos móveis e navegadores.
003	Banco de dados relacional PostgreSQL	Oferece robustez, integridade transacional e flexibilidade para modelagem de dados complexos.
004	Autenticação via OAuth 2.0 / OpenID Connect	Garante segurança, compatibilidade com provedores externos e conformidade com padrões modernos.
005	Adoção de arquitetura em camadas (MVC)	Facilita a separação de responsabilidades, melhora a manutenibilidade e permite escalabilidade modular.
007	Uso de DTOs para entrada e saída de dados	Evita exposição direta de entidades, melhora a segurança e facilita validação de dados.

9. Diagrama de Domínio