

INSTITUTO DE INFORMÁTICA

ARQUITETURA DE SOFTWARE

**LEONARDO RIBEIRO OLIVEIRA PALMEIRA**

**THÂMARA CORDEIRO DE CASTRO**

**LUIZ FELIPE PIRES**

## **PROJETO FINAL:**

Documentação Arquitetura de Software

**LEONARDO RIBEIRO OLIVEIRA PALMEIRA**

**THÂMARA CORDEIRO DE CASTRO**

**LUIZ FELIPE PIRES**

## **PROJETO FINAL:**

Documentação Arquitetura de Software

Desenvolver o documento referente a arquitetura do software de gestão de academia - Haltere.

Professor: JACSON RODRIGUES BARBOSA

**1. Introdução**

**1.1 Finalidade**

O objetivo deste documento é descrever a arquitetura do software de gestão de academia, fornecendo uma visão geral da estrutura, componentes e interações do sistema. Ele serve como um guia para desenvolvedores, fornecendo diretrizes para o projeto, implementação e manutenção da aplicação.

**1.2 Escopo**

O software de gestão de academia é uma aplicação web projetada para auxiliar na administração e operação de academias. Ele oferece funcionalidades para criar e gerenciar treinamentos personalizados, cadastrar e buscar usuários, calcular o Índice de Massa Corporal (IMC), agendar treinamentos, registrar histórico de treinamentos, gerar relatórios de desempenho e gerenciar planos de pagamento. A arquitetura do software é baseada na linguagem de programação Elixir e no framework Phoenix para o Back-end, utilizando o banco de dados PostgreSQL para armazenar os dados. Para o Front-End, será utilizado JavaScript, HTML e CSS, no framework ReactJS, com integração através da GraphQL.

**1.3 Definições, Acrônimos e Abreviações**

* MVC: Model-View-Controller, padrão de arquitetura de software que separa as preocupações relacionadas ao modelo de dados, interface de usuário e lógica de controle.
* Elixir: Linguagem de programação funcional, concorrente e de propósito geral.
* Phoenix: Framework web para desenvolvimento em Elixir.
* Ecto: Biblioteca de mapeamento objeto-relacional (ORM) para banco de dados em Elixir.
* CSV: Comma-Separated Values, formato de arquivo de texto que representa dados tabulares, separando os valores por vírgulas.
* Front-End: Camada de um aplicativo ou sistema web que lida com a apresentação e interação com o usuário. Envolve o desenvolvimento de interfaces de usuário utilizando tecnologias como HTML, CSS e JavaScript.
* JavaScript: Linguagem de programação de alto nível, orientada a objetos e interpretada pelos navegadores web.
* HTML: HyperText Markup Language (Linguagem de Marcação de Hipertexto) é a linguagem padrão para a criação e estruturação de páginas web.
* CSS: Cascading Style Sheets (Folhas de Estilo em Cascata) é uma linguagem de estilo utilizada para definir a aparência e o layout de elementos HTML em uma página web.
* ReactJS: Biblioteca JavaScript de código aberto para construção de interfaces de usuário.
* Framework: Conjunto de ferramentas, bibliotecas e padrões que oferecem uma estrutura para facilitar o desenvolvimento de software.
* PostgreSQL: Sistema de Gerenciamento de Banco de Dados Relacional de código aberto.
* GraphQL: Linguagem de consulta e manipulação de dados para APIs, que oferece uma maneira mais eficiente e flexível de interagir com os serviços de back-end.

**1.4 Visão Geral**

A arquitetura do software de gestão de academia é projetada com uma abordagem modular e escalável, o que possibilita a inclusão e atualização de funcionalidades de forma flexível e adaptável às necessidades em constante evolução. O sistema é baseado em um modelo cliente-servidor, onde os clientes (ou usuários) enviam solicitações ao servidor central. O servidor é responsável por processar essas solicitações, executar as operações necessárias e armazenar os dados relevantes em um banco de dados dedicado.

Essa arquitetura cliente-servidor proporciona uma separação clara de responsabilidades, onde os clientes (aplicações ou interfaces utilizadas pelos usuários) se comunicam com o servidor para obter acesso aos recursos e funcionalidades disponíveis. O servidor, por sua vez, gerencia a lógica de negócios, processando as requisições dos clientes e interagindo com o banco de dados para armazenar e recuperar informações.

Ao adotar uma abordagem modular, o software de gestão de academia é composto por módulos independentes e intercambiáveis, cada um responsável por uma funcionalidade específica. Isso facilita a manutenção e atualização do sistema, permitindo que novas funcionalidades sejam incorporadas sem afetar negativamente as já existentes. Além disso, a escalabilidade da arquitetura possibilita que o software possa lidar com um aumento no número de usuários e demanda de recursos sem comprometer o desempenho.

O software utiliza a linguagem de programação Elixir juntamente com o framework Phoenix para o desenvolvimento web. Além de JavaScript, Html e CSS em paralelo com NodeJS, com integração através da GraphQL. Por fim, o Ecto será utilizado como ORM para a comunicação com o banco de dados PostgreSQL.

O sistema é organizado de acordo com o padrão MVC. Os principais módulos do sistema incluem Controle de Treinamentos, Controle de Usuários, Cálculo de IMC, Teste de Usuário, Aplicação Principal, Exercício, Repositório, Treinamento, Usuário, Instrutor e Administrador. Cada módulo possui responsabilidades específicas e utiliza as tecnologias mencionadas.

O software apresenta um conjunto de requisitos funcionais que especificam as funcionalidades que ele deve oferecer. Esses requisitos incluem:

* Criação de treinamentos personalizados: O sistema deve permitir que os treinadores ou administradores criem treinamentos personalizados para os clientes da academia, adaptando-os às necessidades individuais de cada usuário.
* Criação de novos usuários: O software deve permitir que os administradores adicionem novos usuários ao sistema, incluindo informações como nome, idade, contato, entre outros dados relevantes.
* Cálculo automatizado do IMC (Índice de Massa Corporal): O sistema deve ser capaz de calcular automaticamente o IMC com base nos dados de altura e peso fornecidos pelos usuários.
* Agendamento de treinamentos: Os usuários devem poder agendar horários para suas sessões de treinamento, de acordo com a disponibilidade de horários e instrutores.
* Registro de histórico: O software deve manter um registro detalhado das atividades e interações dos usuários, como treinos realizados, frequência de participação e progresso ao longo do tempo.
* Geração de relatórios de desempenho: O sistema deve fornecer a capacidade de gerar relatórios que apresentem informações sobre o desempenho dos usuários, como resultados de avaliações físicas, objetivos alcançados e métricas de progresso.
* Gerenciamento de planos de pagamento: O software deve permitir que os administradores gerenciam os planos de pagamento dos usuários, definindo valores, prazos e formas de pagamento.

Além dos requisitos funcionais, foram identificados atributos de qualidade importantes para o software:

* Desempenho: O sistema deve ser responsivo e eficiente, garantindo que as operações sejam executadas em tempo hábil, mesmo em momentos de alto tráfego e carga de trabalho.
* Segurança: Deve ser implementada uma camada robusta de segurança para proteger os dados dos usuários, evitar acesso não autorizado e prevenir possíveis vulnerabilidades.
* Usabilidade: A interface do software deve ser intuitiva e amigável, permitindo que os usuários interajam facilmente com o sistema, mesmo sem experiência técnica avançada.
* Escalabilidade: O software precisa ser capaz de lidar com o crescimento do número de usuários e volume de dados sem comprometer o desempenho, garantindo que continue funcionando de forma eficaz à medida que a demanda aumenta.

**2. Representação Arquitetural**

O software de gestão de academia adota uma arquitetura seguindo o padrão MVC (Model-View-Controller), que proporciona uma separação clara e organizada das responsabilidades no sistema. Nessa arquitetura, a lógica de negócios, a apresentação dos dados e o controle das interações são mantidos em módulos distintos, tornando o desenvolvimento e a manutenção mais eficientes.

A abordagem modular e escalável da arquitetura permite que novas funcionalidades possam ser adicionadas e atualizadas sem interferir no funcionamento do restante do sistema. Cada componente do MVC é independente, o que facilita a manutenção e evolução do software ao longo do tempo.

A arquitetura é baseada no modelo cliente-servidor, onde o servidor atua como o ponto central de processamento e armazenamento dos dados. Os clientes do sistema (aplicativo web) acessam o servidor para enviar solicitações, como a criação de novos usuários, agendamento de treinamentos ou geração de relatórios. O servidor, por sua vez, processa essas solicitações, realiza as operações necessárias no Model e retorna os resultados apropriados para serem apresentados na View.

A interface do usuário é implementada como um aplicativo web, o que permite que administradores da academia, instrutores e usuários finais acessem o sistema de qualquer dispositivo com acesso à internet. Essa abordagem torna o software mais acessível e facilita a distribuição de informações relevantes para os diversos usuários envolvidos na gestão da academia.

**3. Metas e Restrições da Arquitetura**

3.1 Usabilidade:

* O Elixir e o framework Phoenix são conhecidos por sua capacidade de criar aplicativos web com uma sintaxe concisa, fornecendo uma interface intuitiva e amigável para todos os tipos de usuários, incluindo administradores, instrutores e usuários finais da academia.
* Com Elixir e Phoenix é possível criar arquiteturas de software escaláveis e orientadas a eventos. Isso facilita a definição de fluxos de trabalho eficientes e bem projetados, garantindo que as ações e funcionalidades do software sejam executadas de forma suave e sem interrupções, garantindo que os fluxos de trabalho sejam bem projetados e que as ações sejam executadas de forma eficiente.
* HTML, CSS e JavaScript fornece ferramentas para criação de interfaces responsivas, consequentemente, permitindo uma experiência consistente em diferentes dispositivos, como desktops, tablets e smartphones.

3.2 Confiabilidade:

* Elixir é conhecida por sua escalabilidade e tolerância a falhas, o que significa que o software terá menos tempo de inatividade tornando a aplicação confiável e estável, minimizando o tempo de inatividade e evitando a perda de dados, garantindo que o software esteja, no mínimo, 95% disponível durante todo o dia.
* Elixir e o Phoenix têm opções de incluir autenticação e autorização adequadas, criptografia de dados em trânsito e em repouso, proteção contra ataques de injeção de código e implementação de políticas de segurança recomendadas.
* O uso de JavaScript, HTML e CSS trará a implementação de boas práticas de segurança do lado do cliente, como validação de entrada de dados e proteção contra ataques de XSS (Cross-Site Scripting) e CSRF (Cross-Site Request Forgery).
* O PostgreSQL suporta recursos como replicação, pontos de recuperação (recovery points) e cópias de segurança incrementais, fortalecendo mecanismos de backup e recuperação de dados.

3.3 Suportabilidade:

* O framework Phoenix, construído em cima de Elixir, facilita a organização do código em componentes reutilizáveis e estimula a criação de uma arquitetura escalável, tornando o desenvolvimento do software em forma modular, permitindo a adição de novas funcionalidades e a atualização de componentes conforme necessário, não sendo exigente quanto a máquina que o acessa.
* JavaScript, HTML e CSS são tecnologias amplamente suportadas e executadas em praticamente todos os navegadores e sistemas operacionais modernos. Essa combinação permite que o software desenvolvido seja acessado em diferentes dispositivos e navegadores, garantindo sua utilização em uma ampla gama de ambientes

3.4 Funcionalidade:

* O software deve fornecer um conjunto abrangente de funcionalidades para atender às necessidades da academia. Isso inclui recursos como gerenciamento de membros, agendamento de aulas, registro de presença, acompanhamento de progresso, emissão de relatórios e integração com sistemas de pagamento.
* As funcionalidades devem ser bem definidas, consistentes e fáceis de usar, permitindo que os usuários realizem suas tarefas de forma eficiente e eficaz.
* Elixir, Phoenix, JavaScript, HTML e CSS são tecnologias que permitem a criação de configurações e opções flexíveis para adaptar o software às necessidades específicas da academia, proporcionando uma experiência personalizada para os usuário com as necessidades específicas da academia.

3.5 Eficiência:

* O software deve ser projetado para ser eficiente em termos de desempenho e recursos. Deve ser responsivo e rápido ao executar tarefas, garantindo tempos de resposta rápidos e processamento eficiente de dados.
* Deve ser otimizado para lidar com grandes volumes de dados e cargas de trabalho simultâneas, sem comprometer o desempenho.
* Também é importante minimizar o consumo de recursos do sistema, como memória e capacidade de armazenamento, para garantir uma utilização eficiente dos recursos do ambiente de hospedagem.
* Para isso, será utilizado Elixir, sendo conhecido por sua eficiência em termos de desempenho e recursos. Ele utiliza uma máquina virtual (BEAM) que executa tarefas de forma concorrente e distribuída, permitindo lidar com grandes volumes de dados e cargas de trabalho simultâneas.

3.6 Manutenibilidade:

* O software será desenvolvido seguindo boas práticas de programação e arquitetura de software, facilitando sua manutenção e evolução.
* Será modular e terá uma estrutura bem organizada, permitindo que os desenvolvedores realizem alterações e correções de forma rápida e eficiente, minimizando o impacto em outras partes do sistema.
* Será acompanhado de uma documentação clara e abrangente, descrevendo sua estrutura, funcionalidades e procedimentos de manutenção. Também deve ser fácil de depurar, testar e integrar com outras ferramentas e sistemas.
* Para isso, Elixir possui uma sintaxe clara e concisa, facilitando a leitura e manutenção do código. Além disso, sua abordagem funcional encoraja a modularidade e o isolamento de componentes, tornando as alterações e correções mais fáceis de serem feitas sem impactar outras partes do sistema.
* O Ecto, como ORM, simplifica a comunicação com o banco de dados e fornece mecanismos de migração de esquema, facilitando a evolução e manutenção do banco de dados.

3.7 Portabilidade:

* O software será projetado para ser portátil e interoperável em diferentes plataformas e ambientes. Será capaz de ser executado em diferentes sistemas operacionais, como Windows, macOS e Linux, sem necessidade de grandes modificações.
* Também deve ser compatível com diferentes bancos de dados, servidores web e serviços de hospedagem, para permitir sua implantação em uma variedade de infraestruturas tecnológicas.
* Além disso, o software deve ser facilmente transferível de um ambiente para outro, permitindo sua migração sem complicações ou perda de dados.

3.8 Restrições de Design:

* O software deve ser desenvolvido utilizando a linguagem de programação Elixir e o framework Phoenix para desenvolvimento web, ademais, sendo utilizado NodeJS com JavaScript, Html e Css para visualização de sua interface funcional.
* O banco de dados utilizado será o PostgreSQL.
* O Ecto será utilizado como ORM (Object-Relational Mapping) para facilitar a comunicação com o banco de dados.
* A arquitetura do software seguirá a abordagem MVC (Model-View-Controller) para organizar os componentes.

3.9 Interfaces:

* O software deve fornecer interfaces adequadas para cada tipo de usuário, com funcionalidades específicas para administradores, instrutores e usuários finais.
* As interfaces devem permitir a realização das principais tarefas, como criação de treinamentos, agendamento de sessões, visualização de relatórios de desempenho, entre outras.
* As interfaces devem ser consistentes em termos de design e usabilidade, proporcionando uma experiência unificada aos usuários.
* Integrações com outros sistemas externos, como sistemas de pagamento, podem ser necessárias e devem ser consideradas na arquitetura.

**4. Visão de Processos**

O sistema de gestão de academia envolve diversos processos que abrangem desde a criação de treinamentos personalizados até a geração de relatórios de desempenho para os usuários. Enfoca os aspectos dinâmicos do sistema e descreve como os processos, threads e tarefas se comunicam e colaboram para realizar as funcionalidades do software. Visa moster como a arquitetura suporta a distribuição e a escalabilidade do sistema.

* Processo de criação de novos usuários:

1. Os administradores podem criar novos usuários no sistema, fornecendo os dados necessários, como nome, idade, gênero, etc.
2. Os dados dos usuários são validados para garantir a integridade e a precisão das informações.
3. Após a validação, os dados dos usuários são inseridos no banco de dados, permitindo o acesso e a gestão das informações.

* Processo de criação de treinamentos personalizados:

1. Os administradores da academia têm a capacidade de criar treinamentos personalizados no sistema vinculados a um usuário.
2. Nesse processo, os administradores definem os parâmetros do treinamento, como exercícios, intensidade, duração, etc.
3. Os dados inseridos são validados para garantir a consistência e a integridade das informações.
4. Uma vez validados, os treinamentos personalizados são armazenados no banco de dados para acesso posterior.

* Processo de acesso às informações detalhadas dos usuários:

1. O sistema fornece informações detalhadas sobre cada usuário, incluindo seus treinamentos associados.
2. Os administradores, instrutores e usuários têm acesso a essas informações, dependendo de suas permissões de acesso.

* Processo de cálculo automatizado do IMC:

1. O sistema realiza o cálculo do Índice de Massa Corporal (IMC) com base em um arquivo CSV fornecido.
2. Os dados do arquivo CSV são processados, e o IMC é calculado com base nas fórmulas apropriadas.
3. O resultado do cálculo é fornecido como uma análise precisa e rápida do IMC do usuário.

* Processo de listagem de treinamentos disponíveis:

1. O sistema permite a listagem de todos os treinamentos disponíveis na academia.
2. Os usuários podem acessar essa lista e selecionar os treinamentos que desejam realizar.

* Processo de agendamento de treinamentos:

1. Os usuários têm a capacidade de agendar treinamentos específicos no sistema.
2. Eles podem selecionar um treinamento disponível e reservar um horário para participar.

* Processo de registro de histórico de treinamentos:

1. O sistema registra o histórico de treinamentos de cada usuário.
2. Esses registros permitem o acompanhamento do progresso e desempenho individual de cada usuário ao longo do tempo.

* Processo de geração de relatórios de desempenho:

1. O sistema é capaz de gerar relatórios de desempenho personalizados para cada usuário.
2. Esses relatórios fornecem métricas relevantes, como frequência de treinamento, progresso em relação aos objetivos, calorias queimadas, etc.
3. Os relatórios podem ser gerados em formato digital ou impresso, conforme necessário.

* Processo de gerenciamento de pagamentos:

1. O sistema permite o registro e o gerenciamento dos pagamentos dos usuários.
2. Os administradores podem registrar os pagamentos recebidos, incluindo informações como data, valor e método de pagamento.
3. Essas informações são armazenadas no banco de dados e podem ser utilizadas para monitorar o status dos pagamentos.

**5. Visões de caso de uso**

—----ANEXO—---

**6. Visão Lógica**

—----ANEXO—---

**7. Visão de Implementação**

—----ANEXO—---

**8. Visão de Implantação**

—----ANEXO—---

**9. Conclusão**

………….

obs: faz isso ->

# **4 Decisões Arquiteturais**

Especificação das decisões arquiteturais tomadas com base nas necessidades levantadas.

# **5 Avaliação Arquitetural**

Análise geral da arquitetural levantada.