22 de novembro de 2024

projeto – sistema web para academia

universidade anhembi morumbi - mooca

São Paulo - SP

documentação a3

Sistemas Distribuídos

Amanda Taynara Dias Cavalcanti

RA: 12523227169

Kaique Adimilson Natividade Barbosa Arraias

RA: 12523225186

Sumário

[Modelo Conceitual e Descritivo do Problema 3](#_Toc183172655)

[Problema Identificado 3](#_Toc183172656)

[Solução 3](#_Toc183172657)

[Finalidades do Site 4](#_Toc183172658)

[Público-Alvo 4](#_Toc183172659)

[Como Será Solucionado? 4](#_Toc183172660)

[Tecnologias Utilizadas 5](#_Toc183172661)

[Back-End 5](#_Toc183172662)

[Front-End 5](#_Toc183172663)

[Ferramentas 5](#_Toc183172664)

[Stored Procedure – (CRUD) - CREATE, READ, UPDATE and DELETE 7](#_Toc183172665)

[Clientes 7](#_Toc183172666)

[Planos 7](#_Toc183172667)

[Assinaturas 8](#_Toc183172668)

[Endereços 8](#_Toc183172669)

[Testes de CRUD 9](#_Toc183172670)

[Banco de dados 12](#_Toc183172671)

[Anexo do script SQL – Criação do Banco de Dados Relacional 12](#_Toc183172672)

[APIs e Conexão com o Banco de Dados 13](#_Toc183172673)

[Como a API se Conecta ao Banco e Funcionamento 13](#_Toc183172674)

[Front-end – WEB 14](#_Toc183172675)

# Modelo Conceitual e Descritivo do Problema

Problema Identificado  
Atualmente, muitas academias enfrentam dificuldades em integrar todos os processos essenciais em um único sistema. A gestão das assinaturas, dos planos e da comunicação com os clientes muitas vezes é feita de forma desconectada, o que leva a diversos desafios, como:

* Uma experiência frustrante para os clientes, que não conseguem entender completamente os serviços e benefícios oferecidos.
* Processos manuais que complicam a gestão de assinaturas, planos e avaliações.
* Falta de opções personalizadas de planos, que atendam realmente às necessidades de cada cliente.

Solução  
O KA-FITCLUB é uma plataforma web pensada para resolver esses problemas, oferecendo uma solução centralizada que facilita a gestão de tudo o que envolve uma academia. Com uma interface moderna e fácil de usar, o sistema permite que os clientes se cadastrem, escolham planos, deixem avaliações sobre a academia e acessem informações personalizadas que a academia os oferecem de maneira prática e eficiente.

**Objetivos**

1. **Automatizar Processos:** Tornar o gerenciamento de assinaturas, planos e avaliações mais ágil e sem erros manuais.
2. **Melhorar a Experiência do Usuário**: Criar uma plataforma intuitiva e acessível para os clientes, permitindo que realizem todo o processo de cadastro e escolha de planos online.
3. **Centralizar Dados**: Garantir que todas as informações sobre o cadastro e assinatura sejam armazenadas de forma segura e organizada em um banco de dados confiável.
4. **Oferecer Serviços Personalizados**: Permitir que os clientes escolham planos que atendam especificamente às suas necessidades, melhorando a experiência geral.
5. **Gerar Relatórios**: Facilitar a coleta de dados e a geração de relatórios para acompanhar o desempenho da academia, com informações como número de assinaturas e feedbacks dos clientes.

## Finalidades do Site

1. **Cadastro de Clientes**: Oferecer um formulário simples e rápido para novos clientes, onde eles poderão fornecer dados pessoais, endereço e o plano desejado.
2. **Gerenciamento de Planos**: Exibir diferentes planos (Básico, Plus e Black) de forma clara, com descrições dos benefícios de cada um, para facilitar a escolha do cliente.
3. **Assinaturas Online**: Permitir que os clientes assinem os planos diretamente pelo site, com todos os dados conectados de maneira eficiente ao banco de dados.
4. **Avaliações**: Oferecer um espaço para que os clientes deixem suas opiniões, através de notas e comentários, ajudando na melhoria contínua dos serviços oferecidos pela academia.

## Público-Alvo

1. **Clientes da Academia**: Pessoas que buscam melhorar sua saúde e bem-estar e desejam uma forma prática de gerenciar suas assinaturas e escolher planos que atendam às suas necessidades.
2. **Administração da Academia**: Profissionais responsáveis pela gestão da academia, que precisam de uma plataforma eficiente para controlar planos, serviços e interagir com os clientes.

## Como Será Solucionado?

1. **Plataforma Web**: O site será intuitivo, responsivo e fácil de usar.
2. **Banco de Dados Relacional**: Usaremos um banco de dados robusto para garantir que todas as informações, desde os dados dos clientes até as avaliações, sejam armazenadas de maneira segura e eficiente.
3. **API Backend**: A plataforma contará com uma API para gerenciar todas as operações de criação, leitura, atualização e exclusão de dados.
4. **Integração entre Frontend e Backend**: O frontend e o backend trabalharão em conjunto para garantir que, ao cadastrar um novo cliente ou associar um plano, todas as informações sejam atualizadas em tempo real no banco de dados.
5. **Acessibilidade e Simplicidade**: A experiência será simples e direta, com um design moderno que facilita a navegação e uso, independentemente do nível de familiaridade com tecnologia.

Tecnologias Utilizadas

## Back-End

* **Node.js**: Plataforma para construir a API.
* **Express.js**: Framework para roteamento e manipulação de requisições.
* **MySQL**: Sistema de gerenciamento de banco de dados.
* **Dotenv**: Gerenciamento de variáveis de ambiente.

## Front-End

* **HTML5**: Estrutura básica do site.
* **CSS3**: Estilização das páginas.
* **JavaScript**: Interatividade e integração com a API.

## Ferramentas

* **DB Designer**: Modelagem do banco de dados.
* **Postman**: Testes das APIs.
* **VS Code**: IDE para desenvolvimento.
* **Git** -Versionamento de código
* **Github** – Disponibilidade do código, visualização de commits e compartilhamento.

Diagrama

Descrição gerada automaticamente**Diagrama Entidade Relacional – DER (3 Forma normal)**

## Stored Procedure – (CRUD) - CREATE, READ, UPDATE and DELETE

## Clientes

**Create POST:**

* Endpoint: POST /api/clientes
* Descrição: Cria um novo cliente no banco de dados com informações pessoais, endereço e plano associado.

**Read GET:**

* Endpoint: GET /api/clientes
* Descrição: Retorna todos os clientes cadastrados.
* Endpoint: GET /api/clientes/:id
* Descrição: Retorna detalhes de um cliente específico pelo ID.

**Update PUT:**

* Endpoint: PUT /api/clientes/:id
* Descrição: Atualiza os dados de um cliente existente.

**Delete DELETE:**

* Endpoint: DELETE /api/clientes/:id
* Descrição: Exclui um cliente do banco de dados.

## Planos

**Create POST**:

* Endpoint: POST /api/planos
* Descrição: Cria um novo plano no sistema.

**Read GET:**

* Endpoint: GET /api/planos
* Descrição: Retorna todos os planos disponíveis.
* Endpoint: GET /api/planos/:id
* Descrição: Retorna informações de um plano específico.

**Update PUT:**

* Endpoint: PUT /api/planos/:id
* Descrição: Atualiza informações de um plano.

**Delete DELETE:**

* Endpoint: DELETE /api/planos/:id
* Descrição: Remove um plano do sistema.

## Assinaturas

**Create POST:**

* Endpoint: POST /api/assinaturas
* Descrição: Cria uma nova assinatura vinculada a um cliente e a um plano.

**Read GET:**

* Endpoint: GET /api/assinaturas
* Descrição: Retorna todas as assinaturas cadastradas.
* Endpoint: GET /api/assinaturas/:id
* Descrição: Retorna detalhes de uma assinatura específica.

**Update PUT:**

* Endpoint: PUT /api/assinaturas/:id
* Descrição: Atualiza os detalhes de uma assinatura.

**Delete DELETE:**

* Endpoint: DELETE /api/assinaturas/:id
* Descrição: Exclui uma assinatura do banco de dados.

## Endereços

**Create POST:**

* Endereço é criado automaticamente ao registrar um cliente.

**Read GET:**

* Endereço é retornado ao consultar informações de um cliente.

**Update PUT:**

* Atualizações podem ser feitas via cliente, com alteração dos dados de endereço.

**Delete DELETE:**

* Endereço será removido automaticamente ao deletar um cliente, por causa da relação com ON DELETE CASCADE.

## Testes de CRUD

Para fazer um teste vou adotar a tabela CLIENTE para que sirva de exemplo, onde nela vamos criar um cliente, em seguida buscar esse cliente no banco pelo ID, atualizar algumas informações desse cliente e deletar o cliente.

Para realizar esses testes vamos utilizar a ferramenta Postman.

URL: http://localhost:3000/api/clientes

Método: POST

Body (JSON):

Texto

Descrição gerada automaticamente

O resultado esperado é:

Texto

Descrição gerada automaticamente

Caso o usuário coloque um número invalido de CPF, ou CPF existente assim como email, a lógica do backend não permite a criação desse usuário.

Aqui certificamos que o cliente foi criado e seus atributos foram adicionados corretamente, gerando o ID 8 para esse cliente.

Interface gráfica do usuário, Texto, Aplicativo, Email

Descrição gerada automaticamente

Método: GET

Tela de celular com aplicativo aberto

Descrição gerada automaticamente

Inserimos o ID gerado no caminho da URL “api/clientes/{id}” para buscar cliente por ID e o resultado esperado é o print acima, com as informações que foram criadas no método GET.

Método: PUT

Texto

Descrição gerada automaticamente

Para atualizar o cliente, colocamos o ID do cliente escolhido na URL, e colocamos as informações que desejam ser atualizadas, neste caso nome, email e telefone.

A resposta esperada será:

Forma

Descrição gerada automaticamente com confiança média

Método: DELETE

Texto

Descrição gerada automaticamente

Para deletar um cliente, selecionamos o método DELETE e na URL colocamos o ID do cliente que desejamos deletar, logo a resposta esperada será “Cliente excluído com sucesso.”

Esses testes garantem que o CRUD está funcional tanto no backend quanto no banco de dados.

# Banco de dados

O banco de dados segue a 3ª Forma Normal (3FN) e contém as seguintes tabelas:

* **CLIENTE**: Armazena informações dos clientes.
* **PLANO**: Contém os planos disponíveis.
* **ASSINATURA**: Relaciona clientes e planos.
* **ENDERECO**: Informações detalhadas de localização.
* **CIDADE** e **ESTADO**: Estrutura hierárquica para endereços.
* **BENEFÍCIO**: Lista de benefícios vinculados aos planos.
* **PLANO\_BENEFÍCIO**: Tabela intermediária entre planos e benefícios.
* **AVALIACAO**: Avaliações feitas pelos clientes.

## Anexo do script SQL – Criação do Banco de Dados Relacional



# APIs e Conexão com o Banco de Dados

A configuração do banco de dados é realizada em um arquivo dedicado (db.js) para garantir a separação de responsabilidades e facilitar a manutenção.

backend/models/db.js

* **Host:** O endereço do servidor do banco de dados.
* **Usuário:** Nome do usuário autorizado a acessar o banco.
* **Senha:** Senha do usuário.
* **Banco de dados:** Nome do banco de dados

## Como a API se Conecta ao Banco e Funcionamento

A API utiliza um pool de conexões fornecido pela biblioteca **mysql2**. Esse pool permite gerenciar múltiplas conexões simultâneas, melhorando a performance em sistemas com alto tráfego.

O arquivo app.js é o controlador de todas as funcionalidades da API, nele iniciamos o servidor, lê as requisições e define as rotas. **Fluxo Geral:**

1. **Requisição**: O cliente faz uma requisição HTTP (GET, POST, etc.).
2. **Middleware**: O middleware processa a requisição.
3. **Rota**: O controle é passado para a rota correspondente.
4. **Resposta**: A rota executa a lógica necessária (CRUD no banco, validações, etc.) e retorna uma resposta.

## Front-end – WEB

O front-end do projeto é responsável pela interface de usuário, permitindo aos usuários interagir com as funcionalidades do sistema, como cadastro, escolha de planos e navegação pelo site.

**Página Inicial - index.html**

A página inicial contém as seguintes seções principais:

1. **Menu Horizontal**: Navegação para outras seções e páginas.
2. **Banner Inicial**: Destaque visual com o slogan do site.
3. **Seções de Conteúdo**:
   * Exploração de Programas.
   * Benefícios de se associar.
   * Planos disponíveis.
4. **Rodapé**: Contém links úteis e informações de contato.

**Página de Cadastro - cadastro.html**

Esta página permite que novos usuários se cadastrem no sistema. Os campos principais incluem:

* Dados Pessoais: Nome, CPF, Senha.
* Endereço: Rua, Número, Bairro, Cidade, Estado, CEP.
* Contato: Email e Telefone.
* Escolha de Plano: Seleção do plano desejado.7

**Integração com a API** é feita na página de cadastro onde o usuário envia os dados do formulário, a API usando fetch fez a integração entre o front-end e o back-end, ligando ao banco de dados.

# Link do projeto no GitHub

https://github.com/KaiqueArraias/Projeto-SITE-ACADEMIA-A3-UAM-MOOCA