Estrutura – Documento de Arquitetura Técnica

Projeto*:* Sistema de Gestão dos Cursos de Capacitação SUKATECH – Programando o Futuro

Guilherme Silva Virgilli
Joyce Beatriz Ferreira da Costa Silva
Mariana Gonçalves Landi
Ubiratan Alves Paniago Filho
Delvo Resende

Versão 1.0
Data: Setembro/2025

1. Introdução

1.1. Objetivo do documento

O Sukatech é um programa do Governo do Estado de Goiás, gerido pela Secretaria de Estado de Ciência, Tecnologia e Inovação (SECTI), que estabelece uma parceria com a Programando o Futuro (OSC - Organização da Sociedade Civil). Sua proposta principal é o recondicionamento e reciclagem de resíduos eletrônicos. Além desse trabalho, há também um eixo de capacitação gratuita de jovens e adultos em tecnologia. Os cursos oferecidos abrangem a Informática Básica, Robótica, Manutenção de Computadores e Celulares.

O documento de Arquitetura Técnica tem a finalidade de criar uma base sólida e padronizada para o projeto de gestão dos Cursos Técnicos:

Definir visão geral do Sistema: Compreender os requisitos de negócios e construir uma solução computadorizada concreta que possa substituir todos os controles em uso atualmente. Gerenciar Cursos, Turmas, Alunos e Instrutores. O documento visa explicar de que forma o sistema de gestão foi implementado, quais tecnologias foram utilizadas e porque foram adotadas para construção do projeto.

Padronizar o desenvolvimento com regras e padrões que outras equipes de desenvolvimento possam entender e incrementar novas funcionalidades sem impactar no funcionamento inicial do sistema.

Identificar entidades básicas, componentes chaves e mostrar como se relacionam. Descrição completa da estrutura de dados, entidades, dados e atributos.

1.2. Escopo do Sistema - Visão Geral

O objetivo deste projeto é desenvolver uma plataforma computadorizada para substituir os controles atuais em planilhas. A solução visa automatizar e centralizar as informações. Facilitando todas as operações necessárias para fazer a gestão da capacitação de alunos do programa Sukatech. A visão geral do sistema inclui a criação, edição, visualização e exclusão das seguintes entidades básicas:

Cursos: Gerenciamento dos cursos oferecidos: nome, descrição, carga horária, instrutor(es) responsáveis.

Turmas: Organização de turmas para cada curso, com controles especificando datas de início, fim, horários e capacidade de alunos.

Alunos: Cadastro e acompanhamento de alunos matriculados, com dados pessoais e status de matrícula.

Instrutores: Cadastro e informações dos instrutores responsáveis pelos cursos.

Para entendimento dos processos e fluxos de negócio do programa Sukatech foram construídos na plataforma FIGMA os protótipos como uma referência visual para a experiência do usuário e para descrever as funcionalidades que serão implementadas, que é uma prévia de como será a navegação e a interação com o sistema.

Link de acesso aos protótipos do projeto na plataforma FIGMA:
Protótipos Sukatech

1.3. Público-Alvo

Gestores, desenvolvedores e stakeholders.

2. Visão Geral da Arquitetura

A arquitetura adotada no projeto Sukatech foi concebida em camadas, com separação clara das responsabilidades de apresentação, lógica de negócios e persistência de dados. Favorece bastante a manutenibilidade, escalabilidade e clareza nas possíveis evoluções do projeto.
É uma combinação de MVC (Model-View-Controller) no nível de aplicação, o que organiza a lógica e a interações do usuário. REST é o padrão de comunicação entre frontend e backend. A API RESTful expõe os recursos principais do sistema (alunos, cursos e instrutores) e utiliza os verbos HTTP (GET, POST, PUT, DELETE) para manipular dados de forma padronizada e interoperável.
A escolha é motivada por:
Padronização - MVC e REST
Escalabilidade - Separação de camadas favorece a evolução futura do sistema, pode-se adicionar módulos sem o comprometimento do core da aplicação.
Integração - A API REST é consumida por diferentes clientes (web, mobile, dashboards), tornando o sistema mais flexível.

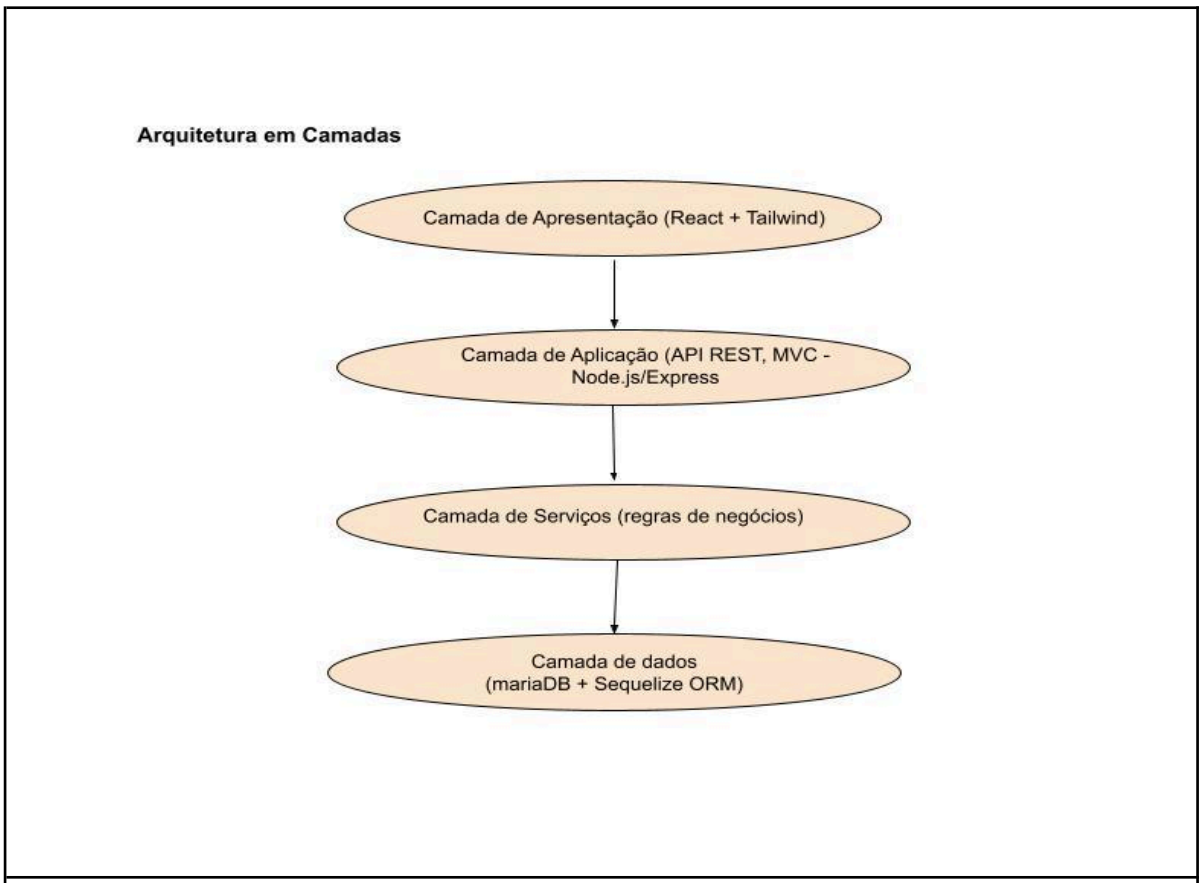

Figura 1. Arquitetura em Camadas

3. Pipeline de Dados e Processos

O fluxo de dados no sistema segue o seguinte ciclo: **Alunos -> Turmas -> Cursos -> Instrutores** estruturados de forma que cada entidade mantenha integridade e rastreabilidade ao longo do processo de capacitação.

Alunos: Cadastrados no sistema, dados são validados e armazenados no banco de dados.
Turmas: Agrupamento de alunos cadastrados com datas definidas, horários e capacidade da turma.
Cursos: Cada curso está associado a turmas e a instrutores responsáveis.
Instrutores: vinculados aos cursos e turmas.

Pipeline de processamento de informações:
Entrada de dados: por meio de formulários web (cadastros, matrículas, criação de cursos, turmas).
Validação e Regras de Negócio: Backend (camada de serviços). Regras: limite de vagas nas turmas,datas válidas e associações corretas entre entidades.
Persistência: Banco Relacional MariaDB, open source, gratuito, dados gravados via ORM (Sequelize), garantindo consistência e integridade.
Consulta e Relatórios: Visualização de dados de forma estruturada (listagens diversas: alunos, turmas, cursos, etc)
Visualização Avançada: dashboards interativos e relatórios que mostram indicadores dos principais resultados dos cursos de capacitação da Sukatech - Programando o Futuro

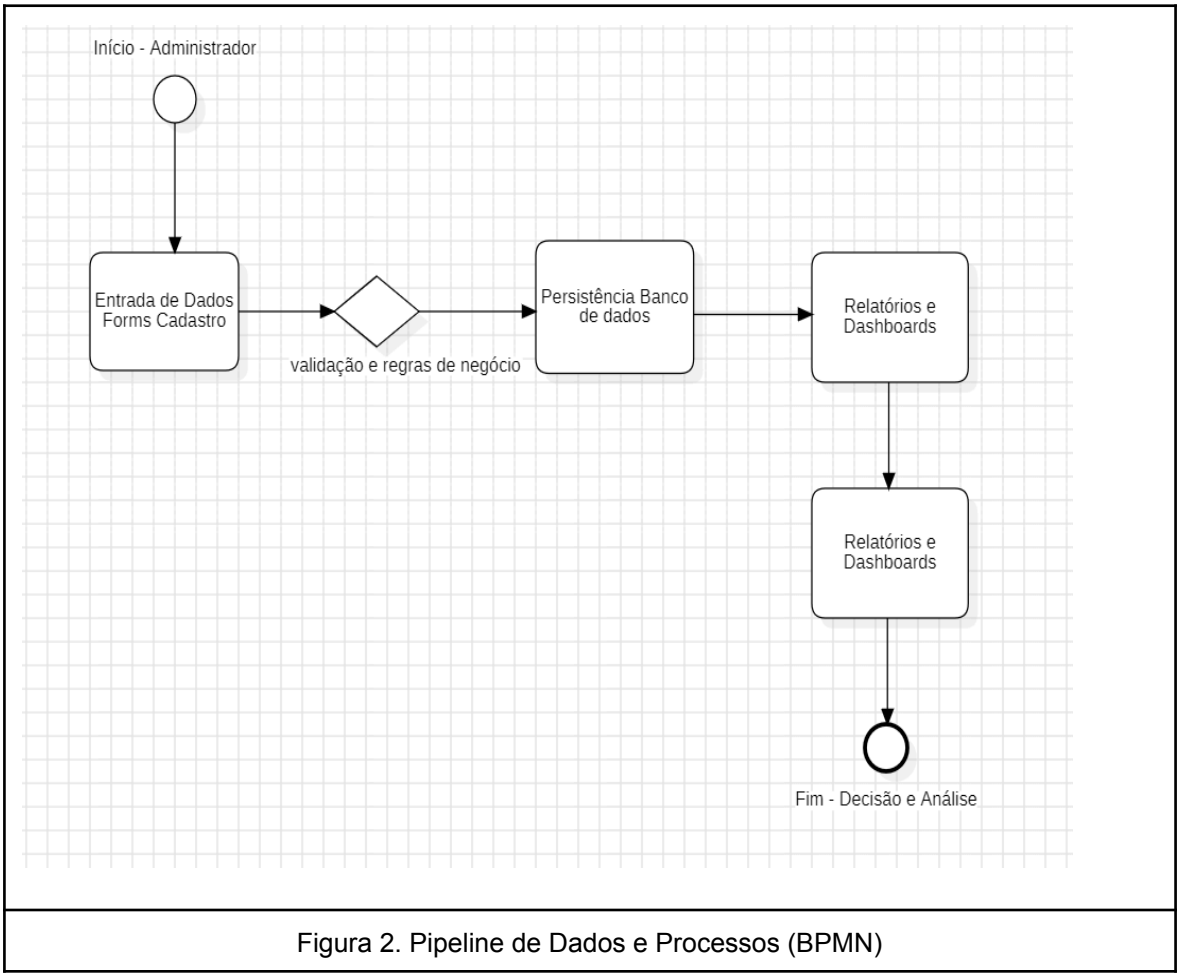

Figura 2. Pipeline de Dados e Processos (BPMN)

4. Stack Tecnológica

4.1. Frontend

O framework escolhido para o desenvolvimento do frontend foi o React, por ser um dos mais consolidados e utilizados no mercado. Ele permite o desenvolvimento de interfaces modernas, escaláveis e de fácil manutenção, além de possuir uma grande comunidade ativa que disponibiliza soluções e boas práticas constantemente.

Entre as principais características que farão diferença na qualidade estão a componentização, que permite a reutilização de componentes, facilita a manutenção e garante consistência visual; o Virtual DOM, que melhora o desempenho da aplicação e torna a interface mais fluida e responsiva; o ecossistema rico, com possibilidade de integração a bibliotecas de roteamento, gerenciamento de estado e estilização; e a escalabilidade, sendo adequado para projetos que podem crescer em funcionalidades ao longo do tempo.

O React fornece flexibilidade e possibilita construir telas dinâmicas com facilidade, o que se ajusta bem ao objetivo do sistema da Sukatech, principalmente na exibição e manipulação dos dados de alunos e cursos.

Apesar da curva de aprendizado intermediária, a ampla documentação, os tutoriais e os exemplos disponíveis reduzem significativamente a barreira de entrada. No contexto de metodologias ágeis, o React se mostra ainda mais vantajoso, pois permite entregas rápidas, incrementais e com alto nível de qualidade.

4.1.1. Estilização

Optou-se pelo uso do Tailwind CSS em conjunto com uma biblioteca de componentes porque ele acelera o processo de criação da interface, reduz a duplicidade de código CSS e garante maior padronização.

O framework oferece utilitários prontos que permitem construir telas responsivas de maneira ágil, além de assegurar consistência e facilidade de bjcustomização.

Quando aliado a bibliotecas de componentes como ShadCN/UI ou Material UI, o Tailwind cobre todas as necessidades de estilização do projeto, incluindo responsividade e acessibilidade, o que garante uma experiência consistente para os usuários e facilita a manutenção futura.

4.1.2. Biblioteca de Componentes

ShadCN/UI Material UI (MUI) Chakra UI

4.2. Backend

A linguagem escolhida para o desenvolvimento do backend é o **TypeScript**, rodando sobre o ambiente **Node.js**.

O TypeScript é um superconjunto de JavaScript que adiciona tipagem estática opcional ao código. Essa característica é fundamental para construir aplicações robustas, pois permite a detecção de erros em tempo de desenvolvimento, melhora a legibilidade do código e facilita a manutenção a longo prazo. Executado sobre o Node.js, um ambiente assíncrono e orientado a eventos, ele se torna ideal para construir APIs de alto desempenho e escaláveis.

Para um sistema de gestão como o SUKATECH, que manipula entidades com regras de negócio bem definidas (Cursos, Turmas, Alunos), a tipagem estática do TypeScript é um grande diferencial. Ela permite a criação de contratos de dados (interfaces e tipos) que garantem a integridade das informações que trafegam pela API, tornando o sistema mais seguro e previsível.

A adoção de TypeScript no backend cria uma sinergia com stacks modernos de frontend (como React, Angular ou Vue), que também o utilizam. Isso permite que a equipe de desenvolvimento compartilhe conhecimento, código (como tipos e interfaces) e ferramentas, o que está perfeitamente alinhado ao objetivo de padronizar o desenvolvimento do projeto.

O TypeScript, apoiado pelo ecossistema gigante do JavaScript e do Node.js, é uma tecnologia madura, com amplo suporte da indústria, uma comunidade global ativa e

documentação extensa. Sua utilização em milhares de sistemas em produção confere a robustez e a confiabilidade necessárias para o projeto SUKATECH.

Atendimento às Regras de Negócio e Comunicação com o Banco de Dados
O ambiente Node.js é altamente eficiente para operações de I/O (entrada e saída), o que inclui a comunicação com bancos de dados. O TypeScript fortalece a implementação das regras de negócio ao garantir que as estruturas de dados estejam corretas. A comunicação com o banco de dados relacional **(seja PostgreSQL, MySQL ou MariaDB)** será otimizada pelo uso de um ORM (Object-Relational Mapping), simplificando as operações de persistência e consulta de dados.

O framework principal para a construção da API será o **expressjs**, e para a camada de acesso a dados, será utilizado o ORM **Sequelize**.

Express.js: Foi escolhido por ser um framework minimalista, flexível e de altíssima performance para Node.js. Ele fornece uma camada essencial de funcionalidades para a construção de APIs, sem impor uma estrutura rígida, o que nos dá liberdade para organizar o projeto conforme as necessidades específicas do SUKATECH.

Sequelize: É um ORM (Object-Relational Mapping) maduro e baseado em Promises para Node.js. Ele simplifica drasticamente a interação com o banco de dados, permitindo mapear as entidades do sistema (Aluno, Turma) para modelos de código. **Sua compatibilidade com os principais SGBDs relacionais do mercado (incluindo PostgreSQL, MySQL e MariaDB)** abstrai a escolha final do banco, garantindo a portabilidade da camada de dados. Além disso, oferece suporte a migrações, validações e gerenciamento de associações complexas.

4.2.1. Padrão Arquitetural (API)

A arquitetura de comunicação adotada será **REST (Representational State Transfer)**.
Descrição do Padrão: A API RESTful expõe os recursos do sistema (alunos, cursos, etc.) através de URLs (endpoints) e utiliza os verbos HTTP (GET, POST, PUT, DELETE) para manipulá-los. É um padrão stateless (sem estado), amplamente consolidado no mercado e de fácil integração com qualquer tipo de cliente.

Detalhes da Arquitetura Backend
Estrutura de Diretórios da Aplicação
Para garantir a organização, manutenibilidade e escalabilidade do código, o backend seguirá o padrão de **Arquitetura em Camadas**. A estrutura de diretórios principal será:
src/

`routes/`: Define todos os endpoints da API e os associa aos seus respectivos controllers.

`controllers/`Responsáveis por receber as requisições HTTP, validar os dados de entrada e orquestrar a resposta.

`services/`: Contém a lógica de negócio principal e complexa do

sistema.

`models/`: Onde os modelos do Sequelize são definidos, representando as tabelas do banco de dados.

`middlewares/`: Funções que interceptam requisições, como para validação de autenticação (JWT).

`config/`: Arquivos de configuração do projeto (banco de dados, variáveis de ambiente).

Segurança e Autenticação

A segurança da API será garantida através de um fluxo de autenticação robusto.

Autenticação de Usuário: A autenticação do administrador será realizada através de um endpoint de login (`/login`) que recebe e-mail e senha.

Armazenamento de Senhas: As senhas nunca serão armazenadas em texto plano. Utilizaremos a biblioteca **bcrypt** para gerar um hash criptográfico de cada senha antes de salvá-la no banco de dados.

Gerenciamento de Sessão com JWT: Após o login bem-sucedido, a API gerará um **JSON Web Token (JWT)**. Este token será enviado ao cliente e deverá ser incluído em todas as requisições subsequentes a rotas protegidas. Um middleware será responsável por validar a autenticidade do JWT a cada requisição.

Qualidade e Padronização de Código

Para assegurar a consistência e a qualidade do código em todo o projeto, serão utilizadas as seguintes ferramentas:

ESLint: Será configurado para analisar o código TypeScript estaticamente, identificando padrões problemáticos, possíveis bugs e garantindo a adesão a boas práticas de programação.

Prettier: Será utilizado para formatar o código automaticamente, garantindo um estilo visual consistente em todos os arquivos e eliminando debates sobre formatação.

Metodologia de Desenvolvimento e Qualidade: TDD (Test-Driven Development)

Para garantir a qualidade e a robustez do software, o desenvolvimento do backend seguirá a metodologia TDD (Desenvolvimento Orientado a Testes).

Ferramentas de Teste: A suíte de testes será construída utilizando o framework **Mocha** como executor de testes (test runner) e a biblioteca **Chai** para as asserções (assertions), uma combinação poderosa e flexível para testes em JavaScript/TypeScript.

Ciclo de Trabalho:

Escrever um Teste (Red): Antes de implementar qualquer funcionalidade, será escrito um teste automatizado que a valide. Inicialmente, este teste falhará.

Implementar a Funcionalidade (Green): O código mínimo necessário será escrito para que o teste passe com sucesso.

Refatorar (Refactor): O código será melhorado e limpo sem alterar seu comportamento, garantindo que o teste continue passando.

Tratamento de Erros e Respostas da API
A API terá um sistema centralizado de tratamento de erros para fornecer respostas consistentes e informativas ao cliente.

Respostas Padronizadas: Serão criados handlers de erro personalizados que retornarão respostas em formato JSON, utilizando os códigos de status HTTP apropriados para cada situação (ex: `400 Bad Request` para dados inválidos, `401 Unauthorized` para falha de autenticação, `404 Not Found` para recursos não encontrados, e `500 Internal Server Error` para erros inesperados).

Documentação da API
Para facilitar a integração com o frontend e o entendimento por parte de outros desenvolvedores, a API será documentada utilizando a especificação **OpenAPI (Swagger)**.

Geração Automática: A documentação será gerada a partir de anotações no próprio código (utilizando bibliotecas como `swagger-jsdoc` e `swagger-ui-express`), garantindo que ela esteja sempre sincronizada com a implementação real da API.

Interface Interativa: O Swagger UI fornecerá uma página web onde será possível visualizar todos os endpoints, seus parâmetros, schemas de dados e até mesmo executar testes diretamente do navegador.

4.3. Banco de Dados

Banco de dados relacional

O sistema gerenciador de banco de dados escolhido para o sistema é o MariaDB. É relacional, código aberto, gratuito e alto desempenho, mantém a estrutura do SGBD MySQL. É compatível com a linguagem SQL para manipulação e gerenciamento dos dados organizados em tabelas. Por se tratar de sistema web e que exigirá armazenar informações de texto, imagem, vídeos, MariaDB atende bem essas necessidades. Apresenta escalabilidade, robustez e diversas funcionalidades que atendem bem às necessidades iniciais do sistema bem como os incrementos que poderão surgir na ampliação do sistema Sukatech.

Modelagem das entidades

É a representação das classes do sistema e as associações entre elas, traduzindo as regras de negócios em um modelo visual. Estão definidos no diagrama os atributos e as operações básicas para manipulação no banco de dados relacional. Traduz também as regras de negócios para persistência das informações. É o plano detalhado para a criação das tabelas: cada classe uma tabela, os atributos são as colunas e os relacionamentos definem as chaves estrangeiras que ligam com as chaves primárias das respectivas tabelas.

Curso: Representa um curso ou treinamento oferecido pelo programa.

Turma: Representa uma instância específica de um curso com alunos.

Aluno: Representa o estudante matriculado em uma turma.

Instrutor: Representa uma pessoa que ministra um curso.

CursoInstrutor: Entidade associativa para representar a associação N:N entre Curso e Instrutor

InstrutorTurma: Entidade associativa para representar a associação N:N entre Turma e Instrutor

O diagrama de classes UML foi construído com a ferramenta StarUML que possibilita criar, visualizar e manter diversos tipos de programas para modelagem de sistemas, como diagramas UML, SysUML e de Sequência. São várias funcionalidades que suportam a modelagem ágil, planejamento de projetos, visualização das funcionalidades de um sistema para usuários e equipes de desenvolvimento.

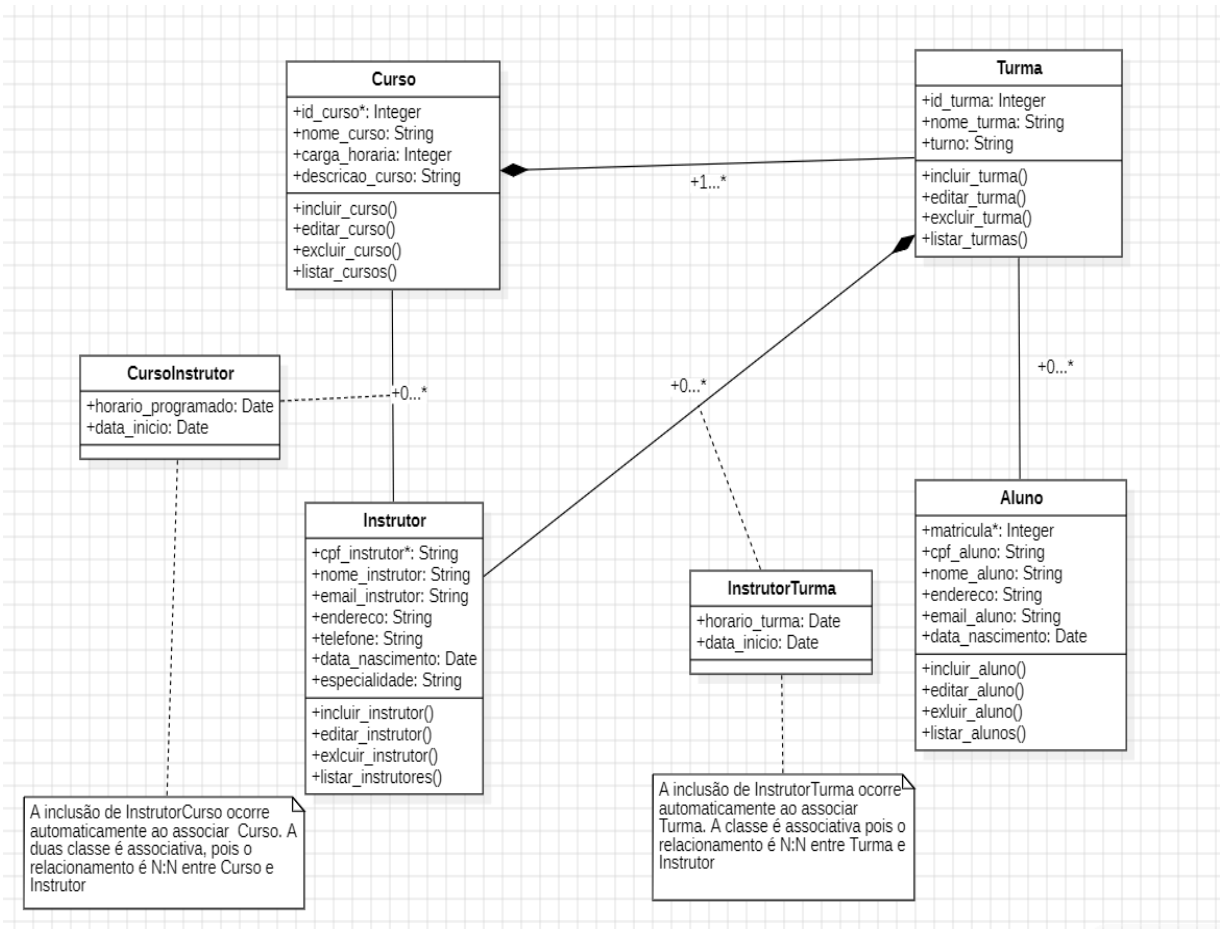

Figura 3. Diagrama de Classes UML

5. Infraestrutura e Diagramas

Representação da estrutura geral do sistema.

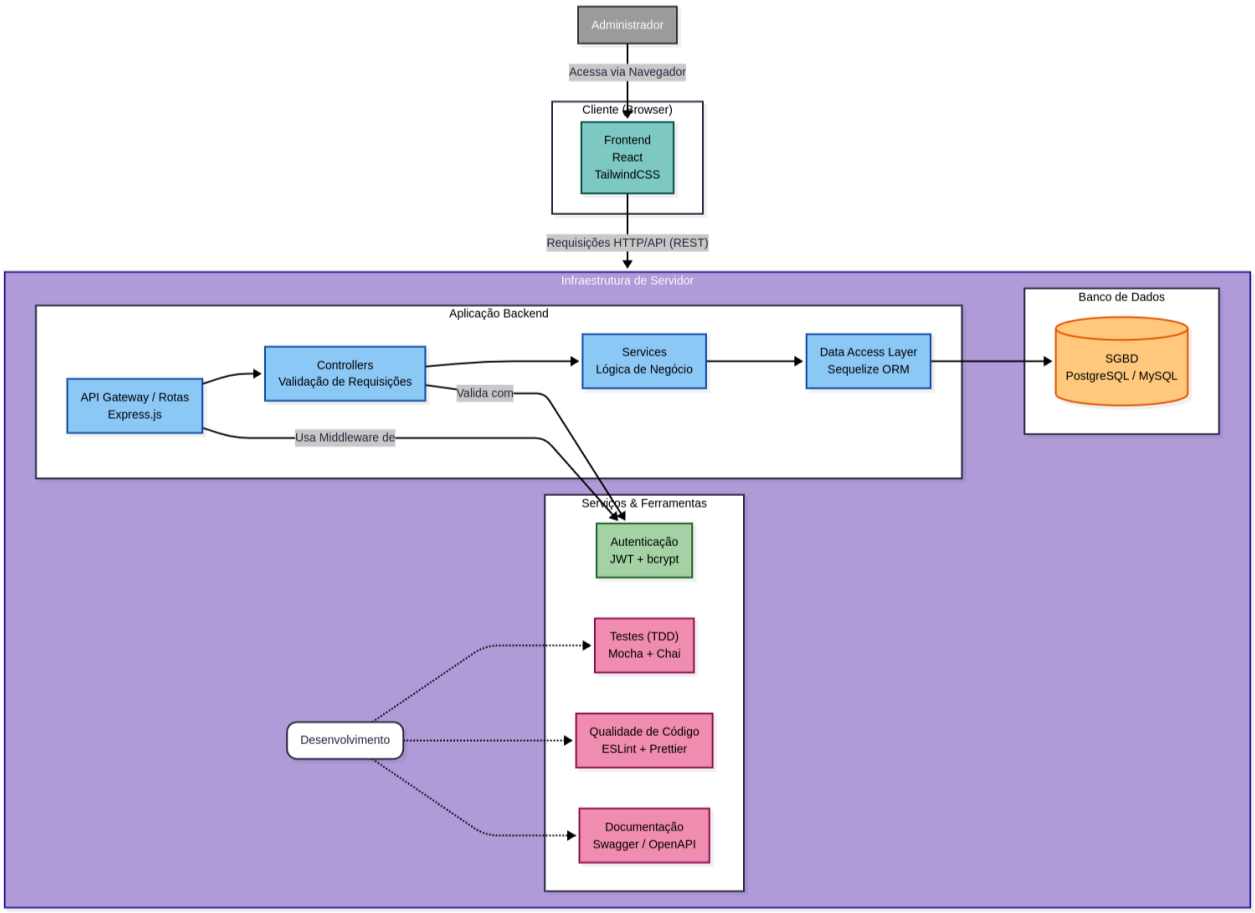

Figura 4. Diagrama de Blocos

5.1. Diagrama de Componentes

Relação entre os módulos internos do sistema (cursos - turmas - alunos - instrutores)
Já temos o diagrama de blocos elaborado, cada um com uma função específica
Interfaces: contratos de interação
Relações: Conexão dos componentes e as interfaces
Estrutura interna: Detalhar o componente que realiza um comportamento externo

5.2. Deployment (decisão futura)

Diagrama de implantação (Deployment) (*definição futura)*

Onde o sistema vai rodar (decisão futura)
Infraestrutura e Implantação: Containerização com Docker
Para padronizar os ambientes de desenvolvimento, testes e produção, o sistema SUKATECH será "containerizado" utilizando **Docker**.
Descrição: O Docker permite empacotar a aplicação e todas as suas dependências (Node.js, bibliotecas, etc.) em um contêiner isolado e portátil.

Benefícios:
Consistência de Ambiente: Garante. que o software funcione da mesma forma em qualquer máquina
Padronização: Facilita a configuração do ambiente para novos desenvolvedores.
Escalabilidade e Portabilidade: Simplifica o processo de implantação (deployment) em qualquer provedor de nuvem.

6. Segurança e autenticação

Perfis de usuário, autenticação JWT, armazenamento seguro de senhas.

7. Considerações Finais

Este documento de Arquitetura de Software tem como objetivo prover uma visão clara e abrangente do Sistema de Gestão Sukatech, proporcionar o entendimento de todos os recursos propostos para serem implementados. A equipe de desenvolvimento têm informações técnicas necessárias para minimizar a possibilidade de erros de interpretação e evitar retrabalhos desnecessários. Auxilia na tomada de decisões futuras, permitindo que alterações ocorram e melhorias sejam implementadas facilmente.