**Documentação da Estrutura da API TILA-API**

Esta documentação detalha a estrutura de diretórios e a função de cada componente principal da API TILA-API, construída utilizando Spring Boot.

**1. Estrutura de Diretórios Principal**

A estrutura do projeto Maven/Spring Boot segue um padrão convencional para facilitar a organização e o desenvolvimento:

Uma imagem contendo Tabela

O conteúdo gerado por IA pode estar incorreto.  
  
**2. Componentes da API**

**2.1. src/main/java/br/com/tila/api/**

Este é o diretório principal do código-fonte Java da aplicação.

* **TilaApiApplication.java**:
  + **Descrição:** É a classe principal da aplicação Spring Boot. Contém o método main() que inicia o servidor embarcado e configura o contexto da aplicação. A anotação @SpringBootApplication consolida @Configuration, @EnableAutoConfiguration e @ComponentScan, facilitando o desenvolvimento de aplicações Spring.
  + **Função:** Ponto de entrada para a execução da API.

**2.2. src/main/java/br/com/tila/api/controllers/**

* **Descrição:** Este pacote contém as classes Controladoras (Controllers) da API. As Controladoras são responsáveis por lidar com as requisições HTTP de entrada, rotear essas requisições para a lógica de negócio apropriada (geralmente serviços), e retornar as respostas HTTP. Elas atuam como a camada de "interface" da API.
* **Arquivos Identificados:**
  + AdminController.java: Gerencia endpoints relacionados a funcionalidades administrativas.
  + CriancaController.java: Gerencia endpoints para operações CRUD (Criar, Ler, Atualizar, Deletar) relacionadas a Crianca.

**2.3. src/main/java/br/com/tila/api/dtos/**

* **Descrição:** Este pacote armazena os Objetos de Transferência de Dados (Data Transfer Objects - DTOs). DTOs são classes simples que encapsulam dados para serem transferidos entre diferentes camadas da aplicação ou entre a aplicação e o cliente (JSON/XML). Eles ajudam a controlar quais dados são expostos e recebidos pelos endpoints da API, desacoplando os modelos de domínio.
* **Arquivos Identificados:**
  + CriancaDTO.java: Usado para representar dados de uma criança em operações da API, possivelmente para entrada (criação/atualização) ou saída (visualização).
  + LoginDTO.java: Encapsula credenciais de login (e-mail/usuário e senha) para autenticação.
  + RegistroDTO.java: Usado para encapsular dados necessários para o registro de um novo usuário (Responsável), como nome, e-mail e senha.

**2.4. src/main/java/br/com/tila/api/models/**

* **Descrição:** Este pacote contém as classes de Modelo (Entities) da aplicação. Essas classes representam as entidades de domínio e são mapeadas para as tabelas no banco de dados através das anotações JPA (Java Persistence API). Elas definem a estrutura dos dados da aplicação.
* **Arquivos Identificados:**
  + Banco.java: Representa informações sobre um banco.
  + Bandeira.java: Representa bandeiras de cartão (Visa, Mastercard, etc.).
  + ConfiguracoesParentais.java: Define configurações ou preferências parentais.
  + Crianca.java: Representa uma criança, provavelmente vinculada a um Responsavel.
  + Jogo.java: Representa informações sobre jogos.
  + Nivel.java: Define níveis dentro do sistema, talvez de jogos ou progresso.
  + Pagamento.java: Representa transações ou detalhes de pagamento.
  + Plano.java: Define os planos de assinatura ou serviço.
  + Progresso.java: Rastreia o progresso de um usuário ou criança no sistema.
  + Recompensa.java: Define recompensas que podem ser obtidas.
  + Responsavel.java: Representa a entidade do responsável (pais, guardiões).
  + SessaoUso.java: Registra o início e fim das sessões de uso da aplicação ou de um jogo.

**2.5. src/main/java/br/com/tila/api/repositories/**

* **Descrição:** Este pacote contém as interfaces de Repositório (Repositories). No Spring Data JPA, essas interfaces estendem JpaRepository (ou similar) e fornecem métodos pré-definidos para operações CRUD básicas e consultas personalizadas ao banco de dados para as entidades correspondentes, minimizando a necessidade de escrever código boilerplate para acesso a dados.
* **Arquivos Identificados:**
  + CriancaRepository.java: Interface para operações de persistência da entidade Crianca.
  + ProgressoRepository.java: Interface para operações de persistência da entidade Progresso.
  + ResponsavelRepository.java: Interface para operações de persistência da entidade Responsavel.

**2.6. src/main/resources/**

Este diretório contém os recursos estáticos e de configuração da aplicação.

* **static/**:
  + **Descrição:** Destinado a arquivos estáticos que serão servidos diretamente pela aplicação, como JavaScript, CSS, imagens, etc.
  + **Função:** Hospedar recursos web que não necessitam de processamento pelo servidor antes de serem entregues ao cliente.
* **templates/**:
  + **Descrição:** Destinado a arquivos de templates que serão processados por um motor de template (ex: Thymeleaf, FreeMarker, JSP) para gerar conteúdo HTML dinâmico.
  + **Função:** Construir páginas web dinâmicas para a interface do usuário (se aplicável, para aplicações web completas, não apenas APIs REST).
* **application.properties**:
  + **Descrição:** O arquivo de configuração principal do Spring Boot. Contém propriedades para configurar o ambiente da aplicação, incluindo a conexão com o banco de dados, portas do servidor, configurações de log, etc.
  + **Função:** Centralizar as configurações da aplicação, permitindo fácil adaptação a diferentes ambientes (desenvolvimento, teste, produção).

**3. Outros Diretórios e Arquivos Relevantes**

* **.mvn/**: Contém wrappers Maven que permitem executar comandos Maven sem precisar instalar o Maven globalmente.
* **src/test/java/**: Diretório para classes de teste Java.
* **target/**: Diretório gerado pelo Maven que contém os artefatos de build (ex: .jar, .war), classes compiladas e outros arquivos temporários.
* **.gitattributes e .gitignore**: Arquivos de configuração do Git para lidar com atributos de arquivos e especificar arquivos/diretórios a serem ignorados pelo controle de versão.
* **HELP.md**: Um arquivo Markdown com informações de ajuda, geralmente gerado por padrão em projetos Spring Initializr.
* **mvnw e mvnw.cmd**: Scripts para executar o Maven Wrapper em sistemas Unix/Linux e Windows, respectivamente.
* **pom.xml**: O Project Object Model (POM) do Maven. Define as configurações do projeto, dependências, plugins de build, e a estrutura geral do ciclo de vida do build.