Changelog: Do Protótipo Avançado para ATV2 - API

Este documento descreve as principais mudanças arquiteturais e funcionais entre o projeto **Protótipo Avançado** e a versão **ATV2 - API**.

1. Visão Geral da Mudança: De Protótipo a Aplicação Full-Stack

A evolução principal foi a transformação de um protótipo de front-end isolado para uma aplicação web completa (full-stack), com uma clara separação entre o cliente (front-end) e o servidor (back-end).

- **Protótipo Avançado**: Uma aplicação **React** que simulava o comportamento de um sistema real. Toda a lógica de negócio e os dados (armazenados em arquivos `.json` na pasta `src/data`) residiam no lado do cliente.
- **ATV2 - API**: Uma arquitetura **cliente-servidor**, onde:
  - O **Front-End (Cliente)**, construído em React, é responsável pela interface do usuário.
  - O **Back-End (Servidor)**, desenvolvido em **Python** com o framework **FastAPI**, gerencia os dados, a lógica de negócio e a comunicação com serviços externos (como a API do Google).

2. Detalhes do Back-End (API com FastAPI)

O coração da nova aplicação é a API RESTful construída com FastAPI.

Estrutura e Endpoints

O arquivo `main.py` define toda a lógica da API. Os principais grupos de endpoints são:

- **Autenticação**:
  - `POST /usuarios`: Cadastra um novo usuário.
  - `POST /login`: Autentica um usuário com email e senha.
  - `POST /auth/google`: Realiza a autenticação via Google, trocando um código de autorização por informações do usuário.
- **Gerenciamento de Conteúdo**:
  - `GET /materias`: Lista todas as matérias.
  - `POST /materias`: Cria uma nova matéria.

  - `GET /simulados/{id_simulado}`: Obtém os detalhes de um simulado específico.

- **Resultados e Desempenho**:
  - `POST /simulados/{simulado_id}/resultados`: Registra o resultado de um simulado feito por um aluno.
  - `GET /simulados/{simulado_id}/resultados`: Lista os resultados de todos os alunos para um simulado específico (visão do professor).
  - `GET /resultados/{usuario_id}`: Lista todos os resultados de um aluno específico.

Modelos de Dados com Pydantic

Para garantir a validação e a consistência dos dados trocados entre o front-end e o back-end, foram utilizados modelos **Pydantic**. Eles definem o "formato" esperado para cada tipo de dado. Exemplos em `main.py`:

- `Usuario`: Define a estrutura de um usuário (id, nome, email, tipo).
- `LoginData`: Espera um `email` e uma `senha` no corpo da requisição de login.
- `Questao`, `SimuladoCompleto`, `Materia`: Modelam a estrutura do conteúdo educacional.

Banco de Dados em Memória

Atualmente, a API utiliza listas e dicionários em Python para armazenar os dados (`db_usuarios`, `db_materias`, etc.). Isso é ideal para desenvolvimento e prototipagem, mas significa que **todos os dados são perdidos quando o servidor é reiniciado**. Em um ambiente de produção, isso seria substituído por um banco de dados persistente (como PostgreSQL, MySQL ou MongoDB).

3. Refatoração do Front-End: Integração com a API

O front-end foi profundamente modificado para se comunicar com o back-end, resultando em uma aplicação mais dinâmica e real.

Exemplo 1: Lógica de Login

A mudança no componente de login é um exemplo claro da transição.

- **Antes (`Prototipo Avançado/src/components/Login.jsx`)**:

  - Os dados dos usuários eram importados de `usuarios.json`.
  - A função `handleLogin` verificava as credenciais comparando com a lista de usuários em memória.
  - O cadastro de novos usuários apenas adicionava a um array no estado do componente.

- **Depois (`ATV2 - API/Front-End/src/components/auth/Login.jsx`)**:
  - A função `handleLogin` agora faz uma requisição `POST` para o endpoint `http://127.0.0.1:8000/login` com o email e a senha.
  - A função `handleCadastro` envia os dados do novo usuário para o endpoint `POST /usuarios`.
  - A autenticação com Google (`handleGoogleLoginSuccess`) envia um código para o endpoint `POST /auth/google`.
  - Após o sucesso, os dados do usuário (ID, tipo, nome) são armazenados no `localStorage` do navegador para manter a sessão ativa.
  - Utiliza o hook `useNavigate` do `react-router-dom` para redirecionar o usuário para o dashboard apropriado.

Exemplo 2: Carregamento de Dados Dinâmicos

O `DashboardAluno` ilustra como os dados que antes eram estáticos agora são carregados dinamicamente.

- **Antes (`Prototipo Avançado/src/components/DashboardAluno.jsx`)**:
  - A lista de `materias` era um array de objetos fixo (hardcoded) dentro do próprio arquivo.
  - O componente simplesmente iterava sobre esse array para renderizar os cards.

- **Depois (`ATV2 - API/Front-End/src/components/aluno/DashboardAluno.jsx`)**:
  - Utiliza o hook `useEffect` para, assim que o componente é montado, fazer uma requisição `GET` para o endpoint `http://127.0.0.1:8000/materias`.
  - Utiliza estados de `isLoading` e `error` para fornecer feedback ao usuário enquanto os dados estão sendo carregados ou em caso de falha na comunicação com a API.
  - Os dados recebidos da API são armazenados no estado `materias`, e o componente é re-renderizado para exibir os cards.

Autenticação e Rotas Protegidas

Foi introduzido o componente `ProtectedRoute.jsx`. Ele "envolve" as rotas que exigem autenticação, verificando no `localStorage` se um usuário está logado e se seu `tipo` permite o acesso àquela página. Isso impede que um usuário não logado acesse os dashboards, por exemplo.

4. Geração de Simulados com IA (Google Gemini)

Esta é a funcionalidade mais inovadora da nova versão.

- **Fluxo**:
  1. Na interface do professor, ele seleciona uma matéria, dá um nome ao novo simulado e faz o **upload de um arquivo** (.pdf ou .txt) contendo o material de estudo.
  2. O front-end envia esses dados para o endpoint `POST /materias/{id_materia}/simulados`.
  3. O back-end recebe o arquivo e extrai seu conteúdo de texto.
  4. Um **prompt** detalhado é montado, instruindo a IA a criar 5 questões de múltipla escolha com base no texto fornecido. O prompt especifica o formato de saída desejado (JSON).
  5. O back-end envia esse prompt para a **API do Google Gemini**.
  6. A resposta da IA (um JSON com as questões) é recebida, validada e armazenada no "banco de dados" em memória.
  7. O novo simulado passa a estar disponível para os alunos.

5. Resumo das Tecnologias e Estrutura

- **Protótipo Avançado**:
  - **Tecnologias**: React, React Router.
  - **Dados**: Arquivos JSON locais.
- **ATV2 - API**:
  - **Frontend**: React, React Router, `@react-oauth/google`.
  - **Backend**: Python 3, FastAPI, Pydantic, Uvicorn (servidor ASGI).
  - **IA**: Google Gemini API.

– **Estrutura**: Monorepo com duas sub-aplicações (`/` para o backend, `/Front-End` para o frontend).