ChatGPT

Análise dos Elementos de Comunicação entre Front-end e Back-end (Gabarit-AI)

Visão geral

O Gabarit-AI é dividido em dois repositórios públicos: `gabarita-ai-frontend` (Next.js) e `gabarita-backend` (Flask). O front-end comunica-se com a API através do serviço `ApiService` e assume que o back-end expõe uma série de endpoints REST em `/api/...`. Durante a análise, foi constatado que o repositório do back-end público contém apenas `run.py`, `main.py`, `render.yaml` e `requirements.txt`, enquanto **os módulos importados em** `main.py` **(**`src/routes/signup`**,** `auth`**,** `planos`**,** `jogos`**,** `news`**,** `opcoes` **etc.) não existem no repositório** [1]. Isso indica uma divergência entre o código implantado no Render e o código versionado no GitHub.

Front-end: Mapeamento de endpoints

A classe `ApiService` define a base da comunicação com o back-end. Ela configura o `baseURL` com a variável de ambiente ou assume `https://gabarita-ai-backend.onrender.com` por padrão [2]. Abaixo segue o mapeamento dos métodos definidos em `services/api.ts` com os endpoints utilizados:

Método da API	Endpoint chamado	Parâmetros Enviados	Observações
`healthCheck()`	`GET /api/health`	–	Ok; endpoint existe em `main.py` [3].
`login(email, password)`	`POST /api/auth/login`	`{ email, password }`	Implementado apenas como *stub*; retorna sempre o mesmo usuário e ignora cadastro [4].
`signup(userData, firebaseToken?)`	`POST /api/auth/cadastro`	Objeto `userData` contendo campos como `name`, `email`, `cpf`, `password`, `cargo`, `bloco` [5]	**Back-end não possui rota** `/api/auth/cadastro` **no repositório público**; erro “Campo nome é obrigatório” indica que a API espera `nome`, não `name`.
`getProfile()`	`GET /api/user/profile`	–	Não existe no código do back-end.
`updateProfile(userData)`	`PUT /api/user/profile`	Objeto parcial de usuário	Não existe no back-end.

Método da API	Endpoint chamado	Parâmetros Enviados	Observações
`generateQuestions(params)`	`POST /api/questoes/gerar`	`{ subject?, difficulty?, count?, bloco?, cargo? }`	Implementado em `main.py` como `gerar_questao_endpoint` [6]; aceita `cargo` e `bloco`, mas também depende de `usuario_id` (não enviado pelo front).
`submitSimulation(answers, questionIds)`	`POST /api/simulados/submit`	`{ answers, questionIds }`	Não existe no back-end.
`getMacetes(id)`, `getPontosCentrais(id)`, `getOutrasExploracoes(id)`	`GET /api/questoes/macetes/:id`, etc.	–	Não existem rotas correspondentes.
`chatDuvidas(id, usuarioId, mensagem)`	`POST /api/questoes/chat-duvidas`	Objeto com `questao_id`, `usuario_id`, `mensagem`	Não há rota.
`getPerformance()`	`GET /api/performance`	–	Não há rota.
`getPlans()`	`GET /api/plans`	–	Provavelmente deveria ser `/api/planos` (Português), conforme `main.py` [7].
`getRanking()`	`GET /api/ranking`	–	Não existe rota; módulo de ranking previsto no blueprint, mas não implementado.
`getNews()`	`GET /api/news`	–	Rota depende do blueprint `news_bp`, mas não presente no repositório.
`createPayment(planId)`	`POST /api/pagamentos/criar`	`{ plano: planId }`	Não existe rota.
`getCargosEBlocos()`	`GET /api/opcoes/cargos-blocos`	–	Repositório não possui rota; em `main.py` há apenas `/api/opcoes/test` [8].
`getBlocosCargos()`	`GET /api/opcoes/blocos-cargos`	–	Não há rota correspondente.

Método da API	Endpoint chamado	Parâmetros Enviados	Observações
`getCargosPorBloco(bloco)`	`GET /api/opcoes/cargos/:bloco`	–	Não existe.
`getBlocosPorCargo(cargo)`	`GET /api/opcoes/blocos/:cargo`	–	Não existe.

Front-end: coleta de dados do usuário

O formulário de cadastro (`app/signup/page.tsx`) define o estado `formData` com as propriedades `name` , `cpf` , `email` , `password` , `confirmPassword` , `cargo` e `bloco` [9] . Ao submeter o formulário, o front-end chama `signup()` passando esse objeto e acrescentando campos adicionais (`level` , `xp` , `accuracy` , `plan`) [10] . O rótulo do campo no formulário está em português (“Nome Completo”) [11] , mas o nome da propriedade é `name` . Esse dado chega ao back-end como `name` e não `nome` .

Back-end: implementação disponível

O arquivo `main.py` registra vários blueprints (signup, auth, questoes, planos, jogos, news, opcoes) e define algumas rotas auxiliares [12] . Todavia, os módulos importados não existem no repositório público, impossibilitando verificar a implementação real. As rotas definidas diretamente em `main.py` são:

- `/api/health` **e** `/health` – Retornam status “healthy” [3] .
- `/api/auth/login` – Retorna um usuário fictício e um token de demonstração [4] .
- `/api/questoes/gerar` – Gera questões com base em `usuario_id` , `cargo` e `bloco` [6] . Se `cargo` ou `bloco` não forem encontrados, devolve erro 404 [13] . O front-end envia `bloco` e `cargo` , mas não `usuario_id` , então o valor padrão `'user-default'` é utilizado.
- `/api/opcoes/test` – Um endpoint de teste para opções, retornando número de cargos carregados [8] .

Não há nenhuma rota visível para `/api/auth/cadastro` , `/api/user/profile` , `/api/opcoes/cargos-blocos` ou `/api/pagamentos/criar` . Logo, as chamadas feitas pelo front-end resultarão em `404 Not Found` ou, quando tratadas por blueprints inexistentes, podem causar erros 500.

Erros e inconsistências identificados

1. **Campo** `name` **vs.** `nome` – O front-end envia `name` no objeto de cadastro, enquanto o back-end aparenta esperar `nome` . O erro exibido (“Campo nome é obrigatório”) decorre dessa discrepância. A interface mostra que o campo está preenchido, mas o backend rejeita porque a propriedade tem outro nome.

2. **Endpoint de cadastro inexistente** – `ApiService` envia os dados para `POST /api/auth/cadastro` [5] . Esse endpoint não aparece em `main.py` nem em qualquer módulo publicado; consequentemente, a requisição retorna erro ou 404. Mesmo que o endpoint exista no código privado, o repositório público não o versionou, dificultando correções colaborativas.

3. **Repositório backend incompleto** – Os módulos `src/routes/signup`, `auth`, `planos`, `jogos`, `news`, `opcoes` e `config/firebase_config` importados em `main.py` não estão presentes no repositório [1]. Essa falta de arquivos pode explicar vários problemas de implantação e impede o diagnóstico preciso das rotas.

4. **Rotas ausentes ou nomenclatura divergente** – Diversos métodos do front-end apontam para rotas que não existem no back-end público, como:

5. `/api/questoes/macetes/:id`, `/api/questoes/pontos-centrais/:id`, `/api/questoes/outras-exploracoes/:id`, `/api/questoes/chat-duvidas` – não implementadas.
6. `/api/performance`, `/api/ranking`, `/api/news`, `/api/user/profile` – não implementadas.
7. `/api/plans` – provavelmente deveria ser `/api/planos` conforme o blueprint de registro dos planos [12].

8. `/api/opcoes/cargos-blocos`, `/api/opcoes/blocos-cargos`, `/api/opcoes/cargos/:bloco`, `/api/opcoes/blocos/:cargo` – não identificadas; o repositório expõe apenas `/api/opcoes/test` [8].

9. **Respostas inconsistentes** – O método `signup()` do `ApiService` espera que o back-end responda com `{ sucesso: true, token: <string>, usuario: <User> }` [14]. Contudo, o stub de login retorna `{ success: True, user: {…}, token: … }` [4]. A diferença nos nomes (`success` vs. `sucesso`, `usuario` vs. `user`) provocará falhas ao interpretar a resposta.

10. **Dependências não configuradas** – O back-end faz referência a `chatgpt_service`, `firebase_config`, MercadoPago e outros serviços externos. Se as chaves de ambiente (`OPENAI_API_KEY`, `FIREBASE_…`, `MERCADO_PAGO_…`) não forem definidas no Render, as importações podem falhar, causando erros no `import` ou na inicialização.

11. **Problemas de implantação** – Em interações anteriores foi observado que:

12. A ausência de `src/__init__.py` causou erro de import ao iniciar o Gunicorn. Adicionar esse arquivo resolveu a importação dos módulos.

13. Configurar o mesmo repositório em “Connect Repository” e “Public Git Repository” no Render gerou conflito, resolvido ao remover a URL duplicada.

14. **Nomenclatura inconsistente de idiomas** – O front-end alterna entre português e inglês (`plans` vs. `planos`, `cargo` vs. `position`, `nome` vs. `name`), o que dificulta a padronização e gera confusão nos nomes das rotas e campos.

15. **Funcionalidades do blueprint ainda não implementadas** – O blueprint de 17 módulos prevê ranking, barra de progresso de vida, monetização via Pix, sistema de afiliados, análise de notícias e FAQ, entre outros. Nenhuma dessas funções tem rotas disponíveis no back-end público, indicando que ainda estão em desenvolvimento ou ausentes.

Sugestão de prompt de correção para Trae (enfatizando erros)

Diagnóstico do projeto Gabarit-AI:

- O formulário de cadastro do front-end envia o campo `name`, mas o back-end rejeita solicitando `nome`. Isso causa o erro “Campo nome é obrigatório”. Verifique se o endpoint `/api/auth/cadastro` está lendo a chave correta ou ajuste o front para enviar `nome`.
- O repositório `gabarita-backend` no GitHub está incompleto: ele importa blueprints (`signup`, `planos`, `news`, `opcoes`, etc.) que não existem no repositório público [1]. Essa ausência impede a validação das rotas e causa erros 404 em diversas chamadas. Confirme se o código implantado no Render corresponde ao repositório público ou se falta publicar arquivos.
- O método de cadastro (`/api/auth/cadastro`) não está implementado em `main.py`. Verifique se ele foi definido em algum módulo privado ou se ainda falta criar a rota.
- As respostas das APIs usam formatos diferentes: alguns retornam `success`, outros `sucesso`, e os objetos de usuário variam (`user` vs. `usuario`). Essa inconsciência gera falhas na interpretação das respostas pelo front-end.
- Várias rotas chamadas pelo front (`/api/questoes/macetes`, `/api/performance`, `/api/ranking`, `/api/news`, `/api/pagamentos/criar`, `/api/opcoes/cargos-blocos` etc.) não existem no back-end, resultando em erros 404. Avalie a lista de endpoints necessários no `ApiService` [15] [16] e implemente as rotas correspondentes no Flask ou ajuste o front.
- A rota `/api/plans` usada pelo front parece incoerente com a nomenclatura do back-end, que usa “planos” em português [7].
- O stub de login retorna dados estáticos e não confere com usuários cadastrados. Será necessário substituir pelo serviço real de autenticação.
- Garanta que o arquivo `src/__init__.py` esteja presente no repositório para permitir importações corretas e que a configuração do Render use apenas “Connect Repository” para evitar conflitos.

Forneça um relatório de como pretende alinhar os nomes de campos e rotas entre front e back, publique o código faltante do backend e padronize as respostas JSON para que o front-end interprete corretamente.

---

[1] [3] [4] [6] [7] [8] [12] [13] main.py
https://github.com/techiaemp-netizen/gabarita-backend/blob/main/src/main.py

[2] [5] [14] [15] [16] api.ts
https://github.com/techiaemp-netizen/gabarita-ai-frontend/blob/master/services/api.ts

[9] [10] [11] raw.githubusercontent.com
https://raw.githubusercontent.com/techiaemp-netizen/gabarita-ai-frontend/master/app/signup/page.tsx