ChatGPT

Checklist técnico para integrar o PortalAluno ao sistema SUBMAX

Este checklist está organizado em seções para auxiliar a equipe de desenvolvimento a conectar o **PortalAluno** ao sistema **SUBMAX**. Cada item contém ações que devem ser executadas em ordem lógica, com base na documentação existente e na estrutura de dados do SUBMAX. Quando possível, itens do checklist são apoiados por trechos da documentação ou do código fonte.

1. Análise da documentação das APIs do SUBMAX

1.1 Endpoints relevantes

API do SUBMAX	Utilidade para o PortalAluno	Referência
/api/alunos/ (AlunoViewSet) – requer `IsAuthenticated`	Permite listar/criar/editar alunos, filtrar por `ativo`, `sexo`, `plano_status` e `plano_tipo`; suporta busca (`search`) nos campos `nome`, `email` e `telefone` e ordenação por atributos como `nome` ou `data_nascimento`. Possui endpoint extra `detalhes` que consolida informações do aluno e calcula a idade, além de endpoints para upload/remover foto (imagem enviada por multipart, convertida para base64 e armazenada em `foto_b64`).	Usar para consultar dados de alunos existentes no SUBMAX. O PortalAluno deve ler apenas os registros do próprio aluno e, se necessário, reutilizar o endpoint `detalhes` para exibir informações consolidadas.
/api/plano/ (PlanoViewSet) – requer `IsAuthenticated`	CRUD básico sobre planos (nome, descrição) com busca e ordenação.	Os planos vinculados ao aluno podem ser exibidos no PortalAluno (ex.: tipo de plano, vigência).
/api/avaliacoes/ (AvaliacaoViewSet) – requer `IsAuthenticated` e escopo por `aluno__id_usuario`	Endpoints para listar, criar e atualizar avaliações físicas. Suporta filtros por `aluno` e `data` e permite upload múltiplo de fotos em `fotos`, armazenadas em JSONField. Há endpoint para servir uma foto específica.	Permite exibir histórico de avaliações físicas do aluno no PortalAluno e, futuramente, fazer upload de novas avaliações.

API do SUBMAX	Utilidade para o PortalAluno	Referência
/api/exercicios/ (ExercicioViewSet) – requer `IsAuthenticated`	CRUD de exercícios. Permite filtrar por `grupo_muscular` ou `grupo_muscular_secundario`, busca por `nome` / `descricao` e ordenação por `nome`. Pode aceitar upload de mídia e grava `id_usuario` do criador.	O PortalAluno poderá listar exercícios para mostrar detalhes de um treino.
/api/grupos-musculares/ (GrupoMuscularViewSet) – requer `IsAuthenticated`	CRUD simples de grupos musculares com busca por `nome` ou `id` e ordenação.	Útil para preencher menus e filtros de grupos musculares no PortalAluno.
/api/agendas/ (AgendaViewSet)	A documentação alerta que a view não define `permission_classes`, podendo estar pública. A agenda representa compromissos/ treinos para um aluno e possui campos como `titulo`, `data`, `hora_inicio`, `hora_fim`, `descricao`, `tipo`, `aluno`, `personal` e `link_externo`. Existe um feed de calendário que filtra por intervalo de datas e aluno/personal e adiciona lembretes da próxima avaliação.	Permite ao PortalAluno exibir o calendário de treinos e avaliações do aluno. A falta de autenticação na API pode exigir tratamento extra de segurança no PortalAluno ou no próprio SUBMAX.
/api/treinos/submaxima/ (SubmaximaAPIView) – requer `IsAuthenticated`	Endpoint de cálculo de submáxima; aceita `POST` com lista `exercicios` e retorna resultados. No código do SUBMAX o método `post` verifica se `exercicios` é uma lista e chama `calcular_submaxima` [1].	Caso o PortalAluno precise calcular cargas submáximas, poderá consumir esse endpoint.

1.2 Modelos e relacionamentos

Modelo	Principais campos / relacionamento	Referência
Aluno	`id_usuario` (usuário proprietário), `organizacao`, `personais` (ManyToMany com `auth.User`), `ativo`, `nome`, `data_nascimento`, `sexo`, `telefone`, `email`, `foto_b64`, `plano_status`, `plano_tipo` (FK para `Plano`), `data_inicio`, `data_fim`, `avaliacao_proxima_em`, `avaliacao_expirada`, além de diversos campos de **anamnese** (condições médicas, estilo de vida, preferências) e campos de **nível de treinabilidade** [2] [3].	Central para o PortalAluno: usado para autenticar (telefone + data de nascimento), exibir dados pessoais e plano, determinar se o aluno está ativo e qual é a próxima avaliação.

Modelo	Principais campos / relacionamento	Referência
Plano	`nome`, `descricao` [4].	Cada aluno pode ter um plano associado (via `plano_tipo`).
Avaliação	Relaciona-se a um `aluno` e armazena dados antropométricos (peso, altura, dobras cutâneas, etc.), cálculos derivados (IMC, RCQ, RCEst, IC) e fotos. Fotos antigas são armazenadas como `ImageField` e as novas em `JSONField` (base64) [5].	Permite exibir histórico de avaliações físicas do aluno no PortalAluno.
GrupoMuscular	Possui `id_usuario`, chave primária `id` (string) e `nome` [6].	Usado por exercícios.
Exercicio	`id_usuario`, `nome`, `descricao`, FK obrigatório `grupo_muscular`, FK opcional `grupo_muscular_secundario`, `equipamento`, `video_url`, `midias` (lista JSON), `imagem_principal`, `gif_animado`, `video_local` [7].	Exercícios compõem séries de treinos.
Agenda	`tipo` (Avaliação ou Treino), `aluno` (FK), `data`, `hora_inicio`, `hora_fim`, `titulo`, `descricao`, `personal` (FK opcional), `link_externo`, `vencido` e campos de criação/modificação [8].	Agenda de compromissos; base para calendário do PortalAluno.
Rotina	Representa plano macro de treinos: `aluno` (FK), `nome`, `data_inicial`, `data_final`, `estrutura` (JSON com semanas, séries e exercícios), `criado_em`, `modificado_em` [9].	Cada rotina define a sequência de treinos.
AccessLinkSemana	Link temporário para acesso móvel a uma semana da rotina; possui `rotina`, `semana_nome`, `semana_indice`, token UUID, `data_expiracao`, `ativo`, timestamps e restrição de unicidade (rotina, semana_indice) [10].	No SUBMAX, progressos são registrados por meio de tokens; pode não ser necessário para PortalAluno se um endpoint próprio for implementado.

Modelo	Principais campos / relacionamento	Referência
ProgressoAlunoSemana	Relaciona um `link` (AccessLinkSemana) a um exercício; possui `id_exercicio`, `series_feitas`, `repeticoes_feitas`, `carga_usada`, `concluido`, `observacao` e `data_registro` [11].	É a tabela usada pelo SUBMAX para registrar progresso no modelo "mobile link". Pode servir de base para a modelagem de progresso do PortalAluno ou inspirar a criação de uma nova tabela.

1.3 Autenticação existente

A maior parte das APIs do SUBMAX utiliza o permission class `IsAuthenticated`, ou seja, espera que o usuário esteja autenticado no contexto do sistema (usuário `auth.User`). Não há, contudo, um endpoint nativo para login de **alunos** baseados em telefone e data de nascimento. A API das agendas é um ponto de atenção, pois a view original não define explicitamente `permission_classes`, sugerindo que pode estar pública. Para o PortalAluno, deverá ser implementado um mecanismo próprio de autenticação de alunos.

2. Integração com o banco de dados

1. **Compartilhar a mesma instância de banco de dados** – o PortalAluno será implantado na mesma máquina que o SUBMAX e deverá utilizar a mesma base de dados. Garanta que a configuração do Django (`DATABASES`) aponte para o mesmo servidor, usuário e schema utilizados pelo SUBMAX.
2. **Reutilizar tabelas existentes** – para evitar duplicação e garantir consistência:
3. A tabela correspondente ao modelo **Aluno** (`apps_alunos_aluno` ou similar) contém telefone, data de nascimento e demais campos necessários [2]; deve ser consultada ao autenticar o aluno.
4. A tabela **Plano** será usada apenas para leitura (nome/descrição)
5. As tabelas **Avaliacao**, **GrupoMuscular**, **Exercicio**, **Rotina** e **Agenda** serão consultadas para exibir avaliações, exercícios, rotinas e compromissos.

6. A tabela **ProgressoAlunoSemana** pode ser reutilizada se o fluxo de progresso for mantido via token, mas provavelmente será melhor criar uma tabela própria (ver ponto 5).

7. **Criar tabelas adicionais** (se necessário) – para registrar progresso de treino de forma mais direta, considere criar um modelo como `ProgressoTreino` com campos `aluno`, `rotina`, `semana_indice`, `exercicio`, `series_feitas`, `repeticoes_feitas`, `carga_usada`, `concluido`, `observacao` e `data_registro`. Usar `unique_together` para evitar duplicidade (`aluno`, `rotina`, `semana_indice`, `exercicio`).

8. **Mapear modelos Django** – importar os modelos existentes do SUBMAX no projeto do PortalAluno. Caso o PortalAluno seja uma aplicação separada, é possível criar modelos "espelhos" com `Meta.managed = False` e `db_table` apontando para as tabelas existentes, evitando que o Django do PortalAluno tente gerenciar (criar/migrar) essas tabelas. Para o novo modelo de progresso, utilize `Meta.managed = True` e migração própria.

9. **Gerenciar migrations com cuidado** – como ambos os sistemas usam o mesmo banco, é crítico evitar conflitos de migração:

10. Isolar migrations do PortalAluno em um diretório diferente ou configurar `MIGRATION_MODULES` para que as migrations de apps compartilhados não sejam executadas novamente.

11. Ao criar novos modelos, verificar se não há nomes de tabela conflitantes com os já existentes.

12. **Tratamento de concorrência** – por compartilhar banco com o SUBMAX, utilize transações atômicas ao registrar progresso ou atualizar dados para evitar condições de corrida. Considere usar `select_for_update` quando for atualizar registros sensíveis.

3. Backend do PortalAluno (Django)

3.1 Configuração inicial

1. **Criar um projeto Django** (separado ou dentro do mesmo repositório) com aplicativo chamado `portal_aluno`.
2. **Configurar** `DATABASES` apontando para a mesma base do SUBMAX. Caso o PortalAluno seja implantado no mesmo projeto Django, reutilizar a configuração existente.
3. **Registrar apps necessários**: `rest_framework` para criação de API, `portal_aluno` e outras libs como `rest_framework.authtoken` ou JWT.
4. **Importar modelos do SUBMAX** utilizando `apps.get_model()` ou definindo modelos com `Meta.managed=False` referenciando as tabelas citadas na seção 2.

3.2 Autenticação do aluno

1. **Endpoint de login** (`POST /portal/login/`):
2. Receber `telefone` e `data_nascimento` no corpo (JSON).
3. Validar se ambos foram fornecidos.
4. Consultar o modelo **Aluno** para encontrar um registro com telefone e data de nascimento correspondentes e `ativo=True` . Usar `select_related('plano_tipo')` para otimizar consultas. Em caso de múltiplos resultados (telefones duplicados), retornar erro e orientar o aluno a contatar o suporte.
5. Criar um **token de autenticação** (pode ser Token DRF ou JWT) e associá-lo ao aluno. Armazenar o ID do aluno no payload do token para identificar o usuário nas próximas requisições.
6. Retornar o token no corpo da resposta.

7. Implementar **limite de tentativas** e log de acessos para prevenir ataques de força bruta. Considere adicionar segundo fator (SMS/WhatsApp) para aumentar a segurança.

8. **Middleware / Permission**: criar uma permission class `IsAluno` que verifica se o usuário autenticado corresponde a um aluno e, opcionalmente, se coincide com o `id_aluno` passado na URL. Evita que um aluno acesse dados de outro.

9. **Endpoint "me"** (`GET /portal/me/`): retornar informações do aluno autenticado (nome, telefone, data de nascimento, sexo, dados do plano, datas de início/fim, próxima avaliação, foto em base64). Pode reaproveitar a serialização do `Aluno` porém removendo campos sensíveis (condições médicas, medicações etc.).

10. **Configurar CORS e segurança**: habilitar CORS apenas para o domínio do frontend e garantir o uso de HTTPS. Limitar tamanho de upload, configurar rate-limit e logs.

3.3 Endpoints de treinos e progresso

1. **Listagem de rotinas / treinos**: criar `GET /portal/treinos/` que retorna as rotinas do aluno. Usar o modelo **Rotina** e serializador simplificado (nome, período, estrutura). O campo `estrutura` é um JSON com semanas, séries e exercícios [9] ; deve ser repassado ao frontend para construção das telas de treino.
2. **Detalhe da rotina**: `GET /portal/treinos/{rotina_id}/`. Carregar a rotina apenas se `rotina.aluno_id == request.user.aluno_id`. Incluir informações sobre o plano, progresso registrado (ver item 3) e todos os exercícios referenciados. Para cada exercício, buscar nome, imagem e vídeo de `Exercicio` [7] .
3. **Registro de progresso**: criar `POST /portal/progresso/` ou `POST /portal/treinos/{rotina_id}/progresso/`:
4. Body com lista de objetos contendo `exercicio_id`, `series_feitas`, `repeticoes_feitas`, `carga_usada`, `concluido` e `observacao`. Este formato se inspira no `ProgressoSemanaSerializer` [12] , facilitando a reutilização.
5. Validar cada item: verificar se o exercício existe, se pertence à rotina/semana e se o aluno tem permissão para atualizá-lo. Utilizar transação atômica para gravar múltiplos registros.
6. Armazenar no modelo `ProgressoTreino` (nova tabela) ou atualizar `ProgressoAlunoSemana` se optar pela estrutura de tokens. Usar `update_or_create` para evitar duplicidade.
7. Retornar IDs atualizados e mensagem de sucesso.
8. **Exibição de avaliações**: endpoint `GET /portal/avaliacoes/` que lista avaliações do aluno ordenadas por data [5] . Incluir métricas básicas (peso, IMC, RCQ) e links para ver fotos.
9. **Calendário**: criar `GET /portal/calendario/` que retorna eventos do aluno baseados na tabela `Agenda` [8] . Filtrar apenas eventos do aluno e tipo “Treino” ou “Avaliação”. Para cada evento, retornar título, data/hora e link para a rotina ou avaliação correspondente. Evitar usar a API de feed público sem autenticação; se usar, aplicar token de autenticação via querystring.
10. **Integração com submáxima (opcional)**: criar wrapper que envia `POST` para `/api/treinos/submaxima/` com lista de exercícios para calcular cargas submáximas [13] .

3.4 Serializers e validações

1. **AlunoSerializer**: definir campos permitidos (nome, data_nascimento, sexo, telefone, email, plano_tipo, data_inicio, data_fim, avaliacao_proxima_em) e métodos para calcular idade (reaproveitar lógica do endpoint `detalhes`). Ocultar campos sensíveis.
2. **RotinaSerializerPortal**: derivado do `RotinaSerializer` existente [14] , mas filtrando campos para o aluno; por exemplo, incluir apenas `nome`, `data_inicial`, `data_final`, `estrutura` e o campo `id`.
3. **ProgressoSerializerPortal**: baseado no `ProgressoSemanaSerializer` [12] , incluir `rotina` e `semana_indice` se necessário. Implementar `validate` para garantir que `series_feitas` / `repeticoes_feitas` sejam strings ou listas e que `carga_usada` seja coerente.
4. **AvaliacaoSerializerPortal**: selecionar campos principais (data, peso, altura, imc, rcq, rces e fotos), formatar decimais com duas casas e esconder campos que não devem ser mostrados.

4. Frontend (React)

4.1 Telas necessárias

1. **Tela de Login** – formulário com campos de telefone (com máscara) e data de nascimento. Submete para `/portal/login/`. Em caso de sucesso, armazena o token no armazenamento local (localStorage) e redireciona para o dashboard.
2. **Dashboard do Aluno** – exibe saudação, foto (base64), plano vigente, data de início/fim e contadores (número de treinos ativos, data da próxima avaliação). Exibe botões para navegar para "Treinos", "Avaliações" e "Calendário".
3. **Lista de Treinos** – obtida de `/portal/treinos/`. Cada item mostra nome, período e botão para ver detalhes. Possibilidade de indicar status de conclusão (quantos exercícios concluídos/quantos faltam).
4. **Detalhe do Treino/Rutina** – mostra as semanas e séries (de `estrutura.semanas`), lista de exercícios com nome, grupo muscular e mídia (imagem/vídeo). Para cada exercício, permitir informar séries realizadas, repetições, carga e marcar como concluído. Botão "Salvar Progresso" chama o endpoint de progresso.
5. **Histórico de Avaliações** – lista avaliações com data, peso, IMC, RCQ, etc. Permitir expandir para ver fotos (usar URLs do endpoint `foto/{key}`).
6. **Calendário** – consumir `/portal/calendario/` e renderizar eventos com FullCalendar ou componente similar. Permitir clicar em um evento para abrir a rotina ou avaliação relacionada.
7. **Página de perfil** (opcional) – permitir ao aluno atualizar e enviar foto de perfil (chamando o endpoint `POST /api/alunos/{id}/foto/` ou endpoint Portal específico).

4.2 Fluxo de autenticação e estado

1. Ao iniciar a aplicação, verificar se há token em `localStorage` e, se houver, chamar `/portal/me/` para validar e preencher o estado global do usuário.
2. Usar React Context ou Redux para armazenar `authToken`, `aluno` e listas de treinos/avaliações. Fornecer hooks (`useAuth`, `useTreinos`) para acessar e atualizar esses estados.
3. Implementar **roteamento protegido**: rotas privadas redirecionam para login caso o token não exista ou tenha expirado. Incluir interceptador de requisições para anexar `Authorization: Bearer <token>`.
4. Persistir sessão: além de armazenar o token, considerar usar `refresh tokens` caso opte por JWT. Prever logout que remove tokens e limpa estado.

4.3 Consumo das APIs

1. Criar serviço de API (ex.: `api.js`) que encapsula chamadas `fetch` ou `axios`. Definir funções para login, obter dados do aluno, listar treinos, salvar progresso, listar avaliações e calendário.
2. Para upload de imagem de perfil ou avaliações, usar `FormData` e cabeçalhos `multipart/form-data`, conforme a API do SUBMAX.
3. Tratar erros de forma amigável: exibir mensagens claras quando login falhar (usuário não encontrado, dados incorretos) ou quando o registro de progresso não puder ser salvo.

5. Registro de Progresso

1. **Estrutura de dados** – cada registro deve indicar: identificador do exercício (`exercicio_id`), séries executadas (`series_feitas`), repetições (`repeticoes_feitas`), carga utilizada (`carga_usada`), flag `concluido` e observação adicional [11] [12].

2. **Persistência** – na nova tabela `ProgressoTreino` (ou adaptando `ProgressoAlunoSemana`):
3. Chave composta ou restrição de unicidade `aluno + rotina + semana_indice + exercicio` para evitar duplicidades.
4. Se existir registro, atualizar; caso contrário, criar novo. Armazenar `data_registro` automaticamente.
5. **Endpoint** – definir URL única (`POST /portal/progresso/`) que recebe lista de progressos e executa gravação em bloco. Seguir padrão do `ProgressoSemanaSerializer` para aproveitar validações [12].
6. **Validações** – garantir que os exercícios informados pertencem à estrutura da rotina daquele aluno; caso contrário, retornar erro. Validar tipos de dados (números inteiros, strings etc.).
7. **Feedback ao usuário** – após salvar, retornar lista de exercícios atualizados e porcentagem de conclusão do treino. Exibir feedback no frontend (ex.: barra de progresso).

6. Pontos críticos / riscos técnicos

1. **Conflito de migrations** – ao compartilhar banco, migrations conflitantes podem corromper a estrutura. Definir `managed = False` para modelos espelhos, versionar novas tabelas do PortalAluno em app separado e revisar cuidadosamente antes de aplicar migrações.
2. **Concorrência e integridade** – alunos e personais podem atualizar dados simultaneamente. Usar transações (`transaction.atomic`) ao registrar progresso e ao atualizar dados sensíveis. Considere isolamento de leitura (`select_for_update`) ao modificar registros compartilhados.
3. **Segurança da autenticação** – login por telefone + data de nascimento é frágil; implementar limites de tentativa, captchas e (se possível) verificação por SMS para evitar acesso indevido. Armazenar tokens de forma segura e protegê-los contra XSS/CSRF.
4. **Exposição de endpoints do SUBMAX** – a API de agendas aparentemente não possui autenticação; ao consumir, adicionar filtros server-side para que o aluno veja apenas seus compromissos. A equipe do SUBMAX deve avaliar a possibilidade de definir `permission_classes` ou proteger a rota.
5. **Versionamento de API** – mudanças no SUBMAX podem quebrar o PortalAluno. Defina contratos claros e versionados (ex.: `/portal/v1/…`) para a API do PortalAluno e monitore mudanças no SUBMAX. Evite depender de endpoints ou campos não documentados.
6. **Impacto no SUBMAX** – consultas adicionais e gravações de progresso aumentarão a carga no banco. Otimizar consultas (select_related, prefetch), usar cache para dados estáticos (grupos musculares, exercícios) e realizar testes de performance. Documentar as interações e validar com a equipe do SUBMAX antes da implantação.

7. Melhores práticas e melhorias arquiteturais

1. **Centralizar autenticação** – considerar a criação de um serviço de identidade comum para ambos os sistemas (PortalAluno e SUBMAX), com suporte a múltiplos tipos de usuário (personal, aluno). Isso simplificaria a gestão de tokens e a revogação de acesso.
2. **Unificar API REST** – ao invés de criar novos endpoints paralelos, avaliar a possibilidade de evoluir a própria API do SUBMAX para contemplar o acesso de alunos. Isso reduz duplicidade de código e evita divergências de regras de negócio.
3. **Documentação contínua** – manter a documentação de endpoints atualizada e acessível, usando ferramentas como Swagger/OpenAPI. Isso facilita a integração e minimiza erros de interpretação.
4. **Testes de integração** – escrever testes que simulam o fluxo completo (login, listar treinos, salvar progresso) usando banco de testes. Verificar se alterações no SUBMAX não quebram o PortalAluno.

5. **Auditoria e logs** – registrar logins, acessos e alterações de progresso. Essa trilha de auditoria ajuda a identificar fraudes e problemas de uso.
6. **Acessibilidade e UX** – garantir que o PortalAluno seja responsivo, com campos claros (ex.: máscara para telefone, calendário para data de nascimento). Providenciar mensagens de erro localizadas em português.

---

[1] [13] views.py
https://github.com/PedroNettoDs/Submax/blob/main/apps/treinos/views.py

[2] [3] [4] models.py
https://github.com/PedroNettoDs/Submax/blob/main/apps/alunos/models.py

[5] models.py
https://github.com/PedroNettoDs/Submax/blob/main/apps/avaliacao/models.py

[6] [7] models.py
https://github.com/PedroNettoDs/Submax/blob/main/apps/exercicios/models.py

[8] models.py
https://github.com/PedroNettoDs/Submax/blob/main/apps/agendas/models.py

[9] [10] [11] models.py
https://github.com/PedroNettoDs/Submax/blob/main/apps/treinos/models.py

[12] [14] serializers.py
https://github.com/PedroNettoDs/Submax/blob/main/apps/treinos/serializers.py