Diagrama de Classes

Autenticação e Perfil do Usuário

Diagrama de classes (definição completa)

> Classes, atributos, métodos e relacionamentos — versão textual UML detalhada.

Passo a passo (fluxos) — cada requisito mapeado por etapas

---

1) Cadastro do Usuário (fluxo principal)

Objetivo: permitir que novo usuário crie conta com email único e senha segura.

1. Usuário abre tela de cadastro e envia: nome, email, senha, confirmaSenha.
2. Sistema chama ValidadorSeguranca.validarEmail(email).
    - Se false → retornar ResultadoCadastro{sucesso=false, mensagem="E-mail inválido", erros=[...]}.
3. Sistema verifica unicidade do e-mail (consulta banco: SELECT * FROM usuarios WHERE email = ?).
    - Se já existir → retornar erro: E-mail já cadastrado.
4. Sistema chama ValidadorSeguranca.validarSenha(senha):
    - Regras aplicadas:
        - mínimo 8 caracteres

        - pelo menos 1 maiúscula
        - pelo menos 1 minúscula
        - pelo menos 1 número
        - pelo menos 1 caractere especial
    - Se falhar → retornar com lista de motivos (ex.: "senha deve ter pelo menos 1 número").

5. Sistema chama ValidadorSeguranca.compararSenhas(senha, confirmaSenha).
    - Se false → retornar erro "confirmação não corresponde".
6. Se todas as validações passarem:
    - Gerar senhaHash = hash(senha, salt) (usando bcrypt/argon2).
    - Criar objeto Usuario e persistir no banco (inserir id, nome, email, senhaHash, statusConta="ATIVA", dataCriacao).
    - Retornar ResultadoCadastro{sucesso=true, mensagem="Cadastro realizado", dados={usuarioId}}.
7. Pós-condição: usuário criado; dependendo da regra, redirecionar para tela de login **ou** criar sessão automaticamente chamando Sessao.iniciarSessao(usuarioId).

Critérios de aceitação mapeados:

- Formulário valida campos (etapas 2,4,5).
- Mensagens claras em cada erro.
- Cadastro bem-sucedido com e-mail único e senha válida.
- Usuário redirecionado para login ou app logado.

---

2) Login do Usuário (fluxo principal)

Objetivo: autenticar usuário com e-mail e senha.

1. Usuário envia email e senha na tela de login.
2. Sistema valida formato do e-mail (ValidadorSeguranca.validarEmail); se inválido, erro imediato.
3. Sistema busca usuário por e-mail (Usuário).
    - Se não encontrado → Resultado Autenticação{sucesso=false, mensagem="E-mail não cadastrado"}.
4. Comparar hash(senha Fornecida) com usuário.senha Hash.
    - Se diferente:
        - Incrementa contador de tentativas (pode ser em Sessao ou coluna tentativasFalhaLogin).
        - Se contador >= 3 → aplicar política (ex.: bloquear conta temporariamente ou exigir captcha).
        - Retornar erro "E-mail ou senha incorretos".
5. Se correspondente:
    - Zerar contador de falhas.
    - Atualizar usuario.ultimoLogin = agora.
    - Criar sessão: sessao = Sessao.iniciarSessao(usuarioId). Pode gerar JWT com expiração ou criar registro de sessão no servidor.
    - Retornar ResultadoAutenticacao{sucesso=true, dados={token, usuarioDTO}}.

6. Pós-condição: usuário autenticado, sessão ativa, redirecionado à tela principal.

Critérios de aceitação mapeados:

- Autenticação só com credenciais válidas.
- Mensagens claras em falha.
- Sessão mantida após login (token ou cookie).

3) Logout do Usuário (fluxo principal)

Objetivo: encerrar sessão e limpar dados locais.

1. Usuário clica em botão logout.
2. Cliente (app) chama endpoint POST /logout com token/identificador de sessão.
3. Backend localiza Sessao correspondente e executa Sessao.encerrarSessao():
    - Marcar ativo = false, dataExpiracao = agora.
    - Invalidar token (se JWT com blacklist, adicionar ou alterar segredo).
4. Limpar dados locais no cliente (cookies, localStorage, cache).
5. Retornar ResultadoOperacao{sucesso=true, mensagem="Logout efetuado"}.
6. Redirecionar para tela de login.

Critérios de aceitação mapeados:

- Botão disponível nas telas principais.
- Dados de sessão limpos.
- Não é possível voltar à sessão sem login novo.

---

4) Visualização de Perfil do Usuário

Objetivo: mostrar nome e e-mail do usuário autenticado.

1. Usuário acessa tela de perfil.
2. Cliente solicita dados ao endpoint protegido GET /perfil incluindo token de sessão.
3. Backend valida token com Sessao.validarToken(token):
    - Se inválido → 401 Unauthorized.
4. Recuperar Usuario por usuarioId (obtido do token).
5. Retornar UsuarioDTO{nome, email, dataCriacao, ultimoLogin}.
6. Cliente exibe nome e e-mail. Oferecer botão Editar e Voltar.
7. Apenas o usuário dono do token pode ver esses dados (controle pelo token/ID).

Critérios de aceitação mapeados:

- Dados carregados corretamente em tempo real.
- Botão para editar/voltar presente.

---

5) Recuperação de Senha (esqueci a senha)

Objetivo: permitir que usuário solicite e redefina senha via e-mail.

Solicitação do link (fluxo)

1. Usuário entra em "Esqueci a senha" e informa email.
2. Sistema verifica se email existe:
    - Se não existir → retornar mensagem genérica (por segurança) ou erro conforme critério: "E-mail não cadastrado".
3. Se existir:
    - Gerar token único, seguro (UUID + HMAC).
    - Criar registro RecuperacaoSenha com usuarioId, email, token, dataCriacao = agora, dataExpiracao = agora + X horas (ex: 2h), usado = false.
    - Enviar e-mail com link: https://app/exibirRedefinirSenha?token=....
    - Retornar confirmação: "Se o e-mail estiver cadastrado, você receberá um link".
4. Logs e proteção contra abuso (rate limit por IP/email).

Redefinição (fluxo)

1. Usuário clica no link → app/website abre formulário com campo novaSenha e confirmaSenha e token.
2. Cliente envia token, novaSenha, confirmaSenha para POST /redefinir-senha.
3. Backend:
    - Buscar RecuperacaoSenha por token.

- Validar existência, usado == false, e dataExpiracao >= agora. Se falhar → erro "token inválido/expirado".
- Validar ValidadorSeguranca.validarSenha(novaSenha).
- Verificar novaSenha == confirmaSenha.
- Se OK: atualizar Usuario.senhaHash = hash(novaSenha), marcar RecuperacaoSenha.usado = true.
- Opcional: invalidar todas as sessões ativas do usuário (Sessao.encerrarSessao() para cada sessão) — boa prática de segurança.

4. Retornar sucesso: "Senha redefinida com sucesso".
5. Pós-condição: usuário pode logar com a nova senha.

Critérios de aceitação mapeados:

- Confirmação do envio ao inserir e-mail válido.
- Link permite criar nova senha.
- Nova senha validada pelo mesmo critério.
- Mensagem de erro se e-mail não cadastrado (ou política de segurança: não revelar existência).

Validações técnicas importantes (detalhes concretos que você pode colar no relatório)

- **Regex de e-mail (exemplo simples):**
  ^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}$
- **Regras de senha (exemplo de regex):**

    - Pelo menos 8 caracteres, 1 maiúscula, 1 minúscula, 1 número, 1 especial:

      ^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@#$%^&+=!(){}[\]\\|;:'",.<>/?_-]).{8,}$

    - Em vez de depender apenas de regex, devolver mensagens específicas (por exemplo: "faltando número").

- **Hash de senha recomendado:** usar bcrypt ou argon2 com salt e custo apropriado.

- **Tokens de recuperação:** usar UUID + HMAC, armazenar hashed token no banco (por segurança) e comparar hashes.

- **Expiração de token:** configurar X = 2 horas por padrão. Registrar dataCriacao e dataExpiracao.

- **Sessões:**

    - Se usar JWT: expiração curta e mecanismo de refresh; invalidar tokens via lista negra se necessário.

    - Se usar sessions server-side: armazenar sessões com idSessao, token, dataExpiracao, e checar ativo.

- **Tentativas de login:** armazenar contador por usuário e/ou por IP; bloquear temporariamente após N tentativas (ex.: 3) por T minutos.

  GERENCIAMENTO DE TREINOS

Contexto geral

O usuário está logado no aplicativo da academia e deseja **criar, visualizar, editar ou excluir seus treinos personalizados**.
Todas as ações passam por uma interface amigável (app ou web), mas por trás, o sistema utiliza classes e regras de negócio bem definidas.

As principais classes envolvidas são:

- Usuario
- Treino
- Exercicio
- TreinoExercicio
- TreinoService (regras de negócio)
- TreinoRepository (salvar e buscar dados)
    - ExercicioRepository (lista de exercícios disponíveis)

GERENCIAMENTO DE TREINOS

Classe: Usuario

Visão do usuário:
É o dono dos treinos. Ele entra na aba “Meus Treinos”, cria, edita, visualiza e exclui seus treinos.

Atributos:

- idUsuario — identificador único do usuário.
- nome — nome completo.
- email — usado para login.
- senha — usada para autenticação.
- listaTreinos — conjunto de todos os treinos criados por esse usuário.

Métodos:

- criarTreino(nome, listaExercicios) — solicita criação de um novo treino.
- visualizarTreinos() — exibe a lista de treinos salvos.
- editarTreino(treinoId, novosDados) — altera nome, exercícios ou ordem.
- excluir Treino(treinoId) — remove o treino selecionado.

Relações:

- 1 Usuario → *n* Treino (um usuário possui vários treinos).

Classe: Treino

Visão do usuário:
Cada item que aparece na lista “Meus Treinos”. O usuário enxerga o nome, data de modificação e pode abrir para ver os exercícios.

Atributos:

- idTreino — identificador do treino.
- usuarioId — indica a quem pertence.
- nome — nome dado pelo usuário (ex: “Treino de Peito Segunda”).
- dataCriacao — data em que o treino foi criado.
- dataUltimaModificacao — data da última edição.
- ativo — indica se está visível ou foi excluído.
- listaExercicios — exercícios que compõem o treino.

Métodos:

- validarNome() — verifica se o nome não está vazio e é único para o usuário.
- adicionarExercicio(exercicio, series, repeticoes, carga) — adiciona um exercício.
- removerExercicio(exercício) — exclui um exercício da lista.
- atualizarOrdemExercicios() — reorganiza os exercícios.
- atualizarDataModificacao() — registra quando houve a última edição.

Relações:

- 1 Treino → *n* TreinoExercicio (um treino tem vários exercícios).
- 1 Treino pertence a 1 Usuario.

Classe: Exercicio

Visão do usuário:
São as opções que aparecem na lista de exercícios pré-definidos para montar o treino (ex: Supino, Agachamento, Flexão).

Atributos:

- idExercicio — identificador do exercício.
- nome — nome do exercício.
- grupoMuscular — grupo trabalhado (peito, costas, pernas).

- descricao — explicação breve.
- equipamento — indica o equipamento necessário (opcional).

Métodos:

- buscarTodos() — retorna todos os exercícios disponíveis.
- filtrarPorGrupoMuscular(grupo) — busca apenas os de determinado grupo.

Relações:

- 1 Exercicio → *n* TreinoExercicio (um mesmo exercício pode estar em vários treinos).

Classe: TreinoExercicio

Visão do usuário:
Cada linha do treino que ele cria — com as informações de séries, repetições e carga.

Atributos:

- idTreinoExercicio — identificador.
- treinoId — treino ao qual pertence.
- exercicioId — referência ao exercício.
- ordem — posição no treino.
- series — número de séries.

- repeticoes — número de repetições.
- carga — peso utilizado (opcional).
- observacao — campo livre para anotações.

Métodos:

- atualizarDetalhes(series, repeticoes, carga) — edita os dados do exercício.
- validarCampos() — garante que os valores sejam válidos (numéricos, positivos, etc.).

Relações:

- 1 TreinoExercicio → 1 Treino.
- 1 TreinoExercicio → 1 Exercicio.

Classe: TreinoService

Visão técnica:
É o “cérebro” do módulo de treinos. Tudo que o usuário faz na interface passa por aqui antes de ir para o banco.

Métodos:

- criarTreino(usuarioId, nome, listaExercicios)
    - Verifica se o nome é único para o usuário.
    - Valida se há pelo menos 1 exercício.

    - Chama o TreinoRepository para salvar.

- listarTreinos(usuarioId)
    - Busca todos os treinos ativos do usuário.
    - Retorna lista ordenada por data de criação ou modificação.
- editarTreino(usuarioId, treinoId, novosDados)
    - Confirma se o treino pertence ao usuário.
    - Atualiza nome, exercícios e data de modificação.
- excluirTreino(usuarioId, treinoId)
    - Confirma propriedade.
    - Chama TreinoRepository.delete() (ou marca como inativo).

Relações:

- 1 TreinoService → usa TreinoRepository
- 1 TreinoService → usa ExercicioRepository

Classe: TreinoRepository

Visão técnica:
Responsável por gravar, editar, buscar e remover os treinos no banco de dados.

Métodos:

- save(treino) — insere um novo treino.

- findByUsuario(usuarioId) — busca todos os treinos de um usuário.
- update(treino) — atualiza dados de um treino.
- delete(treinoId) — exclui treino e exercícios relacionados.

Relações:

- Manipula diretamente Treino e TreinoExercicio.

Classe: ExercicioRepository

Visão técnica:
Fornece os exercícios pré-definidos para o usuário escolher ao montar o treino.

Métodos:

- findAll() — retorna todos os exercícios disponíveis.
- findByGrupoMuscular(grupo) — filtra exercícios por grupo.

Relações entre as classes

Relação	Descrição
Usuario 1 — * Treino	Um usuário pode criar vários treinos.

Treino 1 — * TreinoExercicio	Um treino contém vários exercícios.
Exercicio 1 — * TreinoExercicio	Um exercício pode estar em vários treinos.
TreinoService → manipula → Treino, TreinoExercicio, TreinoRepository, ExercicioRepository	O serviço centraliza todas as regras de negócio.

Tabelas (visão de banco simplificada)

usuarios
| id | nome | email | senha_hash |

exercicios
| id | nome | grupo_muscular | descricao | equipamento |

treinos
| id | usuario_id | nome | data_criacao | data_ultima_modificacao | ativo |

treino_exercicios
| id | treino_id | exercicio_id | ordem | series | repeticoes | carga | observacao |

Feedbacks exibidos ao usuário

Situação	Mensagem

Nome do treino em branco	“O nome do treino é obrigatório.”
Nenhum exercício adicionado	“Adicione pelo menos um exercício.”
Nome duplicado	“Você já possui um treino com esse nome.”
Sucesso ao criar	“Treino criado com sucesso!”
Sucesso ao editar	“Treino atualizado com sucesso.”
Exclusão confirmada	“Treino excluído com sucesso.”
Exclusão cancelada	“Ação cancelada.”

Fluxo geral de interação

1. Usuário cria um treino: preenche o nome, seleciona exercícios, e clica em “Salvar”.
   O TreinoService valida, grava no TreinoRepository, e retorna mensagem de sucesso.

2. Usuário visualiza treinos: o app chama listarTreinos(), que busca apenas os treinos do usuário logado e mostra na tela.

3. Usuário edita um treino: altera dados, confirma e o sistema atualiza as informações, mantendo histórico de modificação.

4. Usuário exclui treino: o app solicita confirmação, e se positivo, TreinoService.excluirTreino() remove o treino e seus vínculos.

5. Resultado final:

    - O usuário tem total controle sobre seus treinos.

    - O sistema garante integridade, segurança e mensagens claras em cada ação.

Resumo da Experiência do Usuário

Ação	Interação principal	Resultado visível
Criar treino	Preenche nome + escolhe exercícios	Novo treino na lista
Visualizar lista	Abre aba “Meus Treinos”	Lista organizada com datas
Editar treino	Altera nome ou exercícios	Lista atualizada com sucesso
Excluir treino	Confirma exclusão	Treino removido da lista

Execução de Treino

Classe: Usuario

Visão do usuário:
Representa o aluno que está utilizando o aplicativo para realizar seu treino.

Atributos:

- idUsuario: identificador único.
- nome: nome do usuário.
- email: login usado para acessar o sistema.
- senha: usada para autenticação.
- listaTreinos: lista com todos os treinos criados pelo usuário.
- historicoTreinos: registro de todas as sessões finalizadas.

Métodos (ações que o usuário realiza):

- selecionarTreino(treinoSelecionado): o usuário escolhe um treino salvo para iniciar.
- iniciarSessao(treino): aciona o início de uma nova sessão de treino.
- visualizarExercicios(): exibe todos os exercícios do treino ativo.
- marcarExercicioConcluido(exercicio): marca o exercício como feito.

- registrarDetalhes(exercicio, peso, repeticoes, series): insere os dados de execução.

- finalizarSessao(): finaliza o treino atual e salva no histórico.

Relações:

- 1 Usuario → *n* Treino

- 1 Usuario → *n* SessaoDeTreino

Classe: Treino

Visão do usuário:
É o treino escolhido na tela — o usuário vê o nome, a lista de exercícios e pode selecioná-lo.

Atributos:

- idTreino

- nomeTreino

- dataCriacao

- dataUltimaModificacao

- listaExercicios: conjunto de exercícios associados.

Métodos:

- getExercicios(): retorna todos os exercícios do treino.

- validarTreino(): verifica se o treino contém pelo menos um exercício antes de iniciar a sessão.

Relações:

- 1 Treino → *n* Exercicio
- 1 Treino pertence a 1 Usuario

Classe: SessaoDeTreino

Visão do usuário:
Representa a sessão em andamento. O usuário vê na tela o tempo de início, os exercícios e vai marcando o progresso.

Atributos:

- idSessao
- dataInicio
- horaInicio
- dataFim
- status: (ativa, finalizada)
- treinoSelecionado: referência ao treino ativo.
- exerciciosConcluidos: lista de exercícios já marcados como feitos.

Métodos:

- iniciarSessao(treino): cria uma nova sessão com o treino escolhido.
- salvarProgresso(): salva em tempo real os exercícios marcados.
- finalizarSessao(): registra data/hora e envia os dados ao histórico.

Relações:

- 1 SessaoDeTreino → 1 Treino
- 1 SessaoDeTreino → *n* RegistroExercicio
- 1 SessaoDeTreino pertence a 1 Usuario

Classe: Exercicio

Visão do usuário:
Cada linha da lista de treino exibida na tela (exemplo: “Supino”, “Agachamento”, “Rosca direta”).

Atributos:

- idExercicio
- nomeExercicio
- descricao
- grupoMuscular
- imagemExemplo
- statusConclusao (true/false)

Métodos:

- marcarConcluido(): muda o status para concluído.
- desmarcarConcluido(): reverte a marcação.
- exibirDetalhes(): mostra instruções e vídeo demonstrativo.

Relações:

- 1 Exercicio pertence a 1 Treino
- 1 Exercicio → *n* RegistroExercicio

Classe: RegistroExercicio

Visão do usuário:
São os campos que ele preenche durante o treino — peso, séries, repetições.

Atributos:

- idRegistro
- pesoUtilizado
- repeticoesFeitas
- seriesFeitas
- dataRegistro

Métodos:

- registrarDetalhes(peso, repeticoes, series): armazena as informações digitadas pelo usuário.
- editarRegistro(): permite modificar durante a sessão.
- salvarRegistro(): persiste os dados no sistema.

Relações:

- 1 RegistroExercicio → 1 SessaoDeTreino
- 1 RegistroExercicio → 1 Exercicio

Classe: HistoricoDeTreinos

Visão do usuário:
Após finalizar a sessão, ele pode consultar seu histórico completo de treinos realizados.

Atributos:

- idHistorico
- listaSessoesConcluidas
- totalTreinosRealizados

Métodos:

- adicionarSessao(sessao): adiciona a sessão finalizada ao histórico.
- exibirHistorico(): mostra as sessões anteriores.

- filtrarPorData(data): permite visualizar apenas treinos de um período específico.

Relações:

- 1 HistoricoDeTreinos → *n* SessaoDeTreino
- 1 HistoricoDeTreinos pertence a 1 Usuario

Fluxo de Interação (Visão combinada: usuário + técnica)

1. O usuário seleciona um treino. O sistema busca no banco de dados (TreinoRepository) e cria uma instância de SessaoDeTreino.
2. Ao iniciar a sessão, são carregados todos os Exercicio do treino.
3. Conforme o usuário marca um exercício como concluído, a classe SessaoDeTreino atualiza a lista exerciciosConcluidos e salva automaticamente.
4. Quando ele adiciona peso, repetições e séries, um novo RegistroExercicio é criado e vinculado à sessão e ao exercício.
5. Ao clicar em “Finalizar”, o método finalizarSessao() da classe SessaoDeTreino grava data e hora e envia a sessão para o HistoricoDeTreinos.
6. O sistema exibe o feedback de sucesso e bloqueia novas edições na sessão.

Histórico e Acompanhamento

AÇÃO 1: Acessar Histórico de Sessões de Treino

Visão do usuário

O usuário abre o aplicativo e entra na aba “Histórico”.

Na tela, ele vê uma lista de sessões de treino concluídas, cada uma mostrando:

- Nome do treino
- Data e hora da realização
- Duração total da sessão

A lista aparece ordenada da mais recente para a mais antiga.

O usuário pode rolar a tela para ver treinos mais antigos.

Se ainda não tiver sessões concluídas, o app mostra:

> “Nenhum treino concluído até o momento.”

Quando o histórico é carregado, tudo é exibido de forma organizada, permitindo que ele toque em uma sessão para ver os detalhes.

O que acontece por trás (visão técnica)

O aplicativo chama:

```
HistoricoService.listarSessoes(usuarioId)
```

O HistoricoService acessa o SessaoRepository para buscar todas as sessões com:

- usuarioId correspondente
- status = CONCLUIDA

Os resultados vêm ordenados por dataFim (da mais recente à mais antiga).

O serviço transforma os dados em uma lista de SessaoResumoDTO, contendo:

- nome do treino
- data da sessão
- duração (calculada: dataFim - dataInicio)

Essa lista é enviada ao app, que exibe as informações na tela.

Classes envolvidas

- Usuario → representa o dono das sessões.
- SessaoTreino → representa uma sessão concluída.
- Treino → usado para obter o nome do treino associado.
- HistoricoService → coordena a listagem e formatação dos dados.

- SessaoRepository → acessa as sessões armazenadas no banco.

Regras de negócio implementadas

- Apenas sessões concluídas são exibidas.
- Listagem ordenada por data decrescente.
- Cada item mostra data, nome e duração.

Resultado final

O usuário visualiza uma lista limpa e completa das sessões de treino finalizadas, com feedback claro mesmo se o histórico estiver vazio.

AÇÃO 2: Visualizar Detalhes da Sessão

Visão do usuário

O usuário toca em uma sessão específica do histórico, por exemplo:

“Treino de Peito — 10/11/2025”

Na tela de detalhes, ele vê:

- Nome do treino
- Data e duração da sessão
- Lista dos exercícios realizados, cada um mostrando:
    - Nome do exercício

- Peso utilizado
- Séries e repetições registradas

Os dados aparecem de forma clara, organizada e somente para leitura — ele não pode editar nada, pois a sessão já foi concluída.

Há também um botão para voltar ao histórico:

“Voltar para Histórico”

O que acontece por trás (visão técnica)

O app chama:

HistoricoService.obterDetalhesSessao(usuarioId, sessaoId)

1.
2. O serviço valida se:
    - A sessão pertence ao usuário.
    - A sessão está com status CONCLUIDA.
3. O SessaoRepository busca:
    - Os dados da sessão (SessaoTreino).
    - A lista de exercícios realizados (SessaoExercicio).
4. Os dados são convertidos em um SessaoDetalhadaDTO, contendo:
    - Informações gerais (data, duração, nome do treino).

    - Lista dos exercícios com seus registros (peso, séries, repetições).

5. O app exibe tudo em uma tela de visualização, sem opção de edição.

Classes envolvidas

- Usuario → dono da sessão.
- SessaoTreino → contém informações principais da sessão.
- SessaoExercicio → armazena os detalhes de cada exercício feito.
- Treino → fornece o nome original do treino.
- HistoricoService → controla a lógica de exibição dos detalhes.
- SessaoRepository → consulta a sessão e seus exercícios.

Regras de negócio implementadas

- O usuário não pode alterar sessões concluídas.
- Devem ser mostrados todos os exercícios e seus dados.
- Navegação fluida entre histórico e detalhes.

Resultado final

O usuário consegue revisar seu treino completo, com dados precisos e organizados, mantendo a integridade dos registros passados.

Relações entre as classes (explicação)

Relação	Descrição
Usuario 1 — * SessaoTreino	Um usuário pode ter várias sessões concluídas.
SessaoTreino 1 — 1 Treino	Cada sessão é baseada em um treino específico.
SessaoTreino 1 — * SessaoExercicio	Uma sessão contém vários exercícios executados.
SessaoExercicio 1 — 1 Exercicio	Cada registro está vinculado a um exercício.
HistoricoService → SessaoRepository, Treino	O serviço coordena a busca e organização dos dados.

Tabelas (visão simples de banco)

Tabela: usuarios
| id | nome | email | senha_hash |

Tabela: treinos
| id | usuario_id | nome | data_criacao |

Tabela: sessoes_treino
| id | usuario_id | treino_id | data_inicio | data_fim | duracao | status |

Tabela: sessoes_exercicios
| id | sessao_id | exercicio_id | peso | series | repeticoes |

Feedbacks visuais que o usuário recebe

Situação	Mensagem
Nenhuma sessão concluída	“Nenhum treino concluído até o momento.”
Histórico carregado	Lista de sessões exibida normalmente.
Sessão selecionada	Detalhes carregados com sucesso.
Erro ao carregar detalhes	“Não foi possível carregar as informações da sessão.”

Resumo da Execução Técnica

- O HistoricoService é o ponto central que coordena listagem e detalhamento.
- O SessaoRepository consulta os dados no banco.
- As classes SessaoTreino e SessaoExercicio representam os objetos persistidos.

- Tudo é filtrado por usuarioId, garantindo privacidade e integridade dos dados.