

Ano letivo 2024/2025

Arquiteturas Aplicacionais

Sistemas Interativos Confiáveis

Grupo 4 - Portal sobre gestão de ginásio virtual

Rui Cerqueira (pg57902)

Guilherme Rio (pg57875)

André Almeida (pg57866)

César Cardoso (pg57870)

Índice

1) Introdução	3
1.1) Motivação	3
1.2) Contextualização	3
2) Modelação	4
2.1) Requisitos	4
2.1.1) Requisitos funcionais	4
2.1.2) Requisitos não funcionais	5
2.2) <i>Use Cases</i>	6
2.3) Modelo de Domínio	7
2.4) Diagrama de classes <i>PIM</i>	8
2.5) Protótipos	9
2.5.1) <i>Home page</i>	9
2.5.2) <i>Workouts List</i>	10
2.5.3) <i>Edit Workout</i>	10
2.5.4) <i>Workout Execution</i>	11
2.5.5) <i>Progress Page</i>	12
2.5.6) <i>Clients Page</i>	12
3) Implementação	13
3.1) Padrão arquitetural e tecnologias utilizadas	13
3.2) Camada de Dados	14
3.2.1) Entidades da Camada de Dados	14
3.2.2) Relações entre Entidades	19
3.3) Camada da lógica de negócio	20
3.3.1) Modelos	20
3.3.2) Serviços	21
3.3.3) <i>Controllers</i>	22
3.4) Camada de apresentação	23
3.4.1) Interfaces desenvolvidas	23
3.4.2) <i>Stores Pinia</i>	32
4) Análise do sistema	34
4.1) Especificações e Capacidade de Escalabilidade da Aplicação	34
4.2) Performance	34
4.2.1) <i>100 threads</i>	35
4.2.2) <i>500 threads</i>	36
4.2.3) <i>1500 threads</i>	37
4.2.4) <i>3000 threads</i>	38
4.2.5) <i>4000 threads</i>	39
4.2.6) Conclusão dos testes de carga	40
4.3) Usabilidade	40
4.3.1) Avaliação Heurística	40
4.3.2) <i>Cognitive Walkthrough</i>	46
5) Conclusão	50

1) Introdução

Neste relatório iremos apresentar o trabalho prático realizado no âmbito das unidades curriculares de Arquiteturas Aplicacionais e Sistemas Interativos Confiáveis, onde iremos descrever o processo de desenvolvimento da aplicação criada, assim como uma análise da usabilidade e performance da aplicação.

O tema escolhido pelo nosso grupo foi o desenvolvimento de uma aplicação para um portal sobre a gestão de um ginásio virtual, onde se pretende que os utilizadores possam realizar a criação de planos de treino, registar a sua execução e verificar o seu progresso. Para os professores, será possível guiar os seus alunos, criando planos de treino para os mesmos e acompanhando o seu progresso.

1.1) Motivação

A criação deste portal online para a gestão de um ginásio virtual serve como uma solução inovadora e necessária para um mundo cada vez mais digital. Esta aplicação pretende facilitar aos utilizadores o acesso aos seus planos de treino, permitindo-lhes não só executá-los de forma autónoma, mas também acompanhar o seu progresso ao longo do tempo. Da mesma forma oferece aos professores uma ferramenta eficaz para gerir os seus alunos, criar planos personalizados para eles e monitorizar o seu desempenho, o que promove uma maior interação entre professor e aluno, e ajuda-os a cumprir os seus objetivos.

1.2) Contextualização

O objetivo da aplicação desenvolvida é facilitar a criação, execução e acompanhamento do progresso dos respetivos utilizadores, oferecendo aos utilizadores todas as funcionalidades necessárias para atender a essas necessidades. As principais funcionalidades da aplicação são as seguintes.

- Criação de uma conta na aplicação.
- Criação de planos de treino por parte de utilizadores normais.
- Criação de planos de treino para clientes por parte de professores.
- Execução e registo de dados de treino do cliente.
- Acompanhamento do progresso pessoal.
- Professores conseguem fazer o acompanhamento do progresso dos clientes.

2) Modelação

Nesta secção vamos mostrar toda a modelação desenvolvida no processo de desenvolvimento da aplicação. Desde a recolha dos requisitos funcionais e não funcionais aos modelos *UML* desenvolvidos e protótipos da interface.

2.1) Requisitos

O levantamento de requisitos é dos mais importantes no processo de desenvolvimento, pois define as funcionalidades da aplicação e como estas devem funcionar.

2.1.1) Requisitos funcionais

Autenticação

- O sistema deve permitir que clientes se registem através de um email e palavra-passe.
- O sistema deve permitir que utilizadores façam *login* com as suas credenciais, email e palavra-passe, para a utilização do sistema.
- O sistema deve garantir que não sejam registados dois emails iguais.

Professor-Aluno

- O sistema deve permitir associar professores a clientes.
- O sistema deve permitir ao professor remover um aluno associado.
- Professores devem poder visualizar os clientes que lhes estão associados.
- Professores devem poder aceder a um painel com informações do cliente.

Planos de treino

- Utilizadores devem poder criar/editar/remover os seus próprios planos de treino.
- Professores devem poder criar/editar/remover planos de treinos para clientes.
- Deve ser possível distinguir planos criados pelo cliente ou pelo professor.
- Deve ser possível definir se um plano de treino tem horário fixo ou livre.
- Se o plano de treino tiver horário fixo, tem de ser selecionado no mínimo um dia da semana.
- O sistema deve permitir ao utilizador ativar e desativar treinos e fazer distinção dos mesmos.

Exercícios

- O sistema deve disponibilizar uma biblioteca com diversos exercícios, cada um com descrição e indicação do grupo muscular que treina.
- O sistema deve permitir filtrar a biblioteca por grupos musculares e tipo.
- Cada exercício de musculação deve conter detalhes de repetições, peso a utilizar e tempo de descanso recomendado.
- Utilizadores devem poder visualizar os dados de um exercício no seu plano de treino.
- O sistema deve permitir aos utilizadores adicionar “notas” nos exercícios de um plano de treino.

Execução de um plano de treino

- Utilizadores devem poder registar os *sets* efetuados para cada exercício com o peso utilizado e repetições realizadas.
- O sistema deve permitir aos utilizadores iniciar o treino, que inicia o seu temporizador.
- O utilizador deve poder editar *sets* já registados.
- O sistema deve permitir aos utilizadores adicionar *sets* durante a realização de um treino.
- Ao terminar um treino, o sistema deve apresentar um sumário ao utilizador e o utilizador deve poder inserir uma nota final como *feedback* do treino.
- Deve ser possível identificar exercícios completados e exercícios não completados.

Histórico de treinos

- O sistema deve permitir ao utilizador e ao seu professor visualizar o seu histórico de treinos.
- Se o utilizador tiver um plano de treino ativo e com horário fixo, deve poder visualizar claramente o seu próximo treino.
- O utilizador deve poder visualizar os treinos que tem agendados para a semana atual.

Notificações

- O sistema deve enviar notificações ao utilizador quando um professor lhe é associado, quando o professor cria um plano de treino para o cliente e quando o professor atualiza o plano de treino do cliente.

Progresso

- O sistema deve disponibilizar uma *dashboard* de progresso ao utilizador.
- A *dashboard* deve poder ser filtrada por períodos de tempo (semana, mês, anual, sem limite).
- A *dashboard* deve conter detalhes da quantidade de treinos realizados, evolução do peso do utilizador, e volume de peso utilizado pelo utilizador.
- A *dashboard* deve conter um gráfico para o utilizador poder verificar o seu progresso de volume de peso.

Definições

- O sistema deve permitir alterar entre unidades imperiais e métricas

2.1.2) Requisitos não funcionais

Aparência

- O sistema deve adaptar-se a diferentes tamanhos de tela e resoluções, como dispositivos móveis, tablets, etc.
- A interface deve seguir um estilo visual consistente.
- O sistema deve fornecer *feedback* claro das ações realizadas pelos utilizadores.

Usabilidade

- A aplicação é compatível com diferentes plataformas e browsers, incluindo de dispositivos móveis e *desktop*.
- A aplicação fornece *feedback* visual claro e imediato ao utilizador em caso de erro ou falha durante um procedimento

Performance

- As operações realizadas pelo utilizador devem ter um tempo de resposta inferior a 5 segundos.
- O sistema deve suportar pelo menos 500 utilizadores em simultâneo sem degradação significativa de performance.
- A aplicação deve ser capaz de escalar horizontalmente de forma elástica para suportar o aumento de utilizadores e volume de processamentos, mantendo o desempenho mas também os custos controlados.

Manutenção e suporte

- O sistema deve ser projetado de forma modular, seguindo os princípios de design de separação de responsabilidades para facilitar a sua manutenção.

Segurança

- O sistema deve garantir a segurança dos dados dos utilizadores, utilizando encriptação para o armazenamento das palavras passas.

Culturais e políticos

- O sistema deve cumprir o *RGPD*.
- A aplicação deve estar disponível em inglês.
- O sistema deve cumprir todas as regulamentações e leis aplicáveis nos países onde opera, incluindo requisitos específicos de setores.

2.2) Use Cases

Nesta secção serão apresentados os atores e uma especificação tabular dos principais casos de uso, de modo a contextualizar e descrever sucintamente as funcionalidades mais importantes do nosso sistema. O seguinte diagrama de *Use Cases* ilustra as funcionalidades mais importantes do sistema

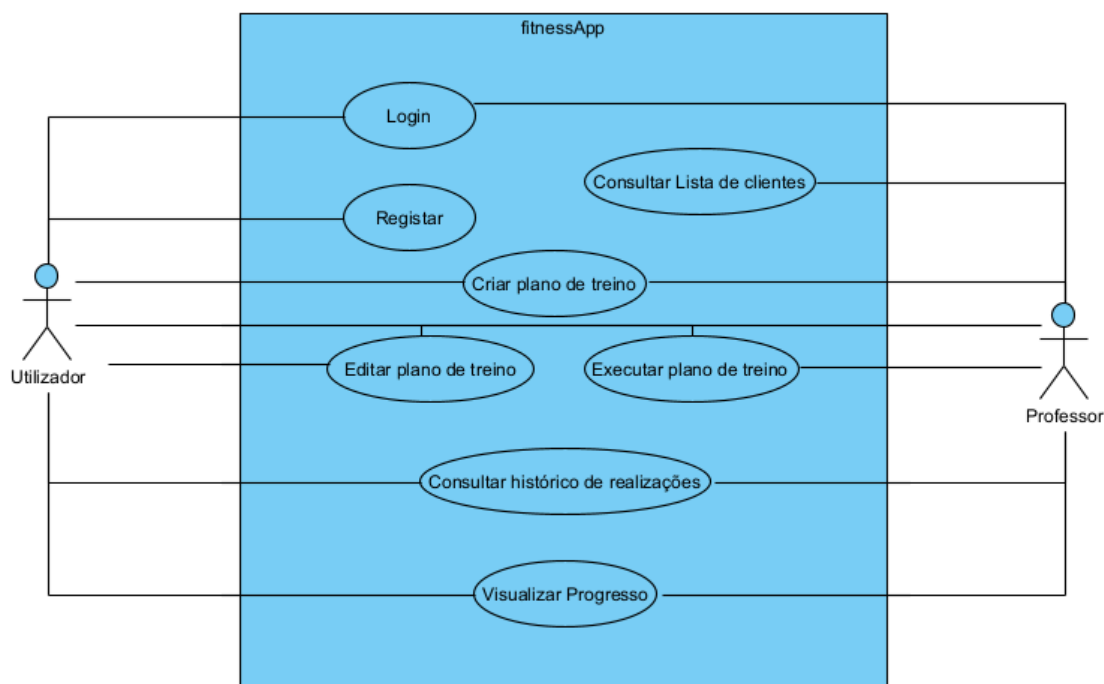


Figura 1: Diagrama de *use cases*

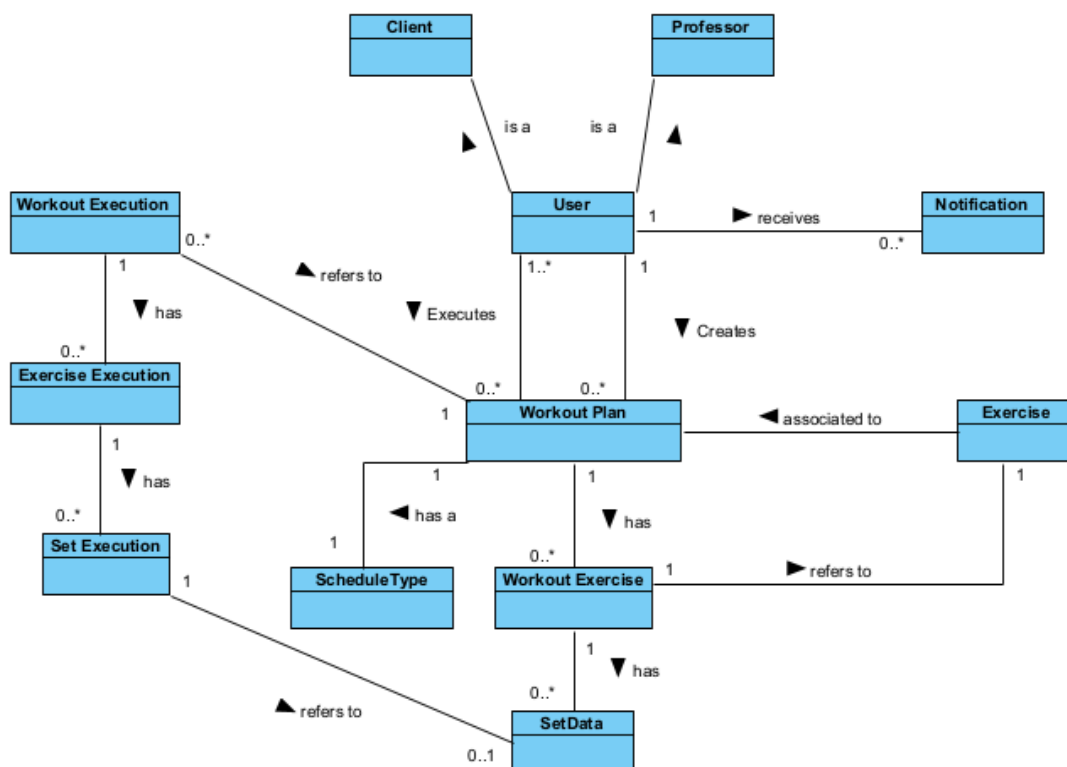
No *use case* “Registrar” o utilizador insere os dados para o identificar no sistema (no caso dos professores, estes serão inseridos na base de dados pelo administrador), e no caso de “Login” o utilizador insere as credenciais de email e palavra-passe.

No *use case* de “Criar plano de treino”, o utilizador apenas tem de seleccionar o nome que pretende dar ao plano de treino. Depois da criação diz respeito ao *use case* de “Editar plano de treino”, onde o utilizador pode alterar detalhes de horário, exercícios e dos *sets* de cada exercício, assim como alterar o nome e a descrição do plano.

Possui também o *use case* de “Executar plano de treino”, que se refere a quando o utilizador insere os dados de realização do seu treino, relativos a cada exercício e *set*. Os dados de cada *set* estão restringidos a valores positivos, sendo também importante de referir que não é possível iniciar um treino se existir outro já em execução.

Por fim temos o *use case* “Visualizar progresso”, que representa o acesso do utilizador a um conjunto de informações acerca do seu progresso, como por exemplo, evolução do seu peso corporal, quantidade de treinos realizados, e volume de peso utilizado total ou por plano de treino. Tudo isto pode ser filtrado por um certo período de tempo.

Com base nos requisitos e *use cases* enunciados, recorreremos ao modelo de domínio para definir todos os conceitos associados ao domínio da aplicação desenvolvida que engloba as principais entidades, as suas relações e as interações que ocorrem na aplicação.



Através do modelo podemos interpretar o funcionamento das relações mais importantes do nosso sistema. De uma forma geral:

- 7

Em relação à entidade *Workout Execution*, esta representa um plano de treino realizado pelo utilizador. Esta entidade está relacionada ao plano de treino realizado, e possui os exercícios realizados e dados sobre os respetivos *sets*.

2.4) Diagrama de classes *PIM*

Com base nos requisitos funcionais e modelo de domínio, desenvolvemos o diagrama de classes *PIM* (*Platform Independent Model*), iniciando o foco na lógica de negócio da aplicação e tendo em atenção os detalhes de implementação mais importantes.

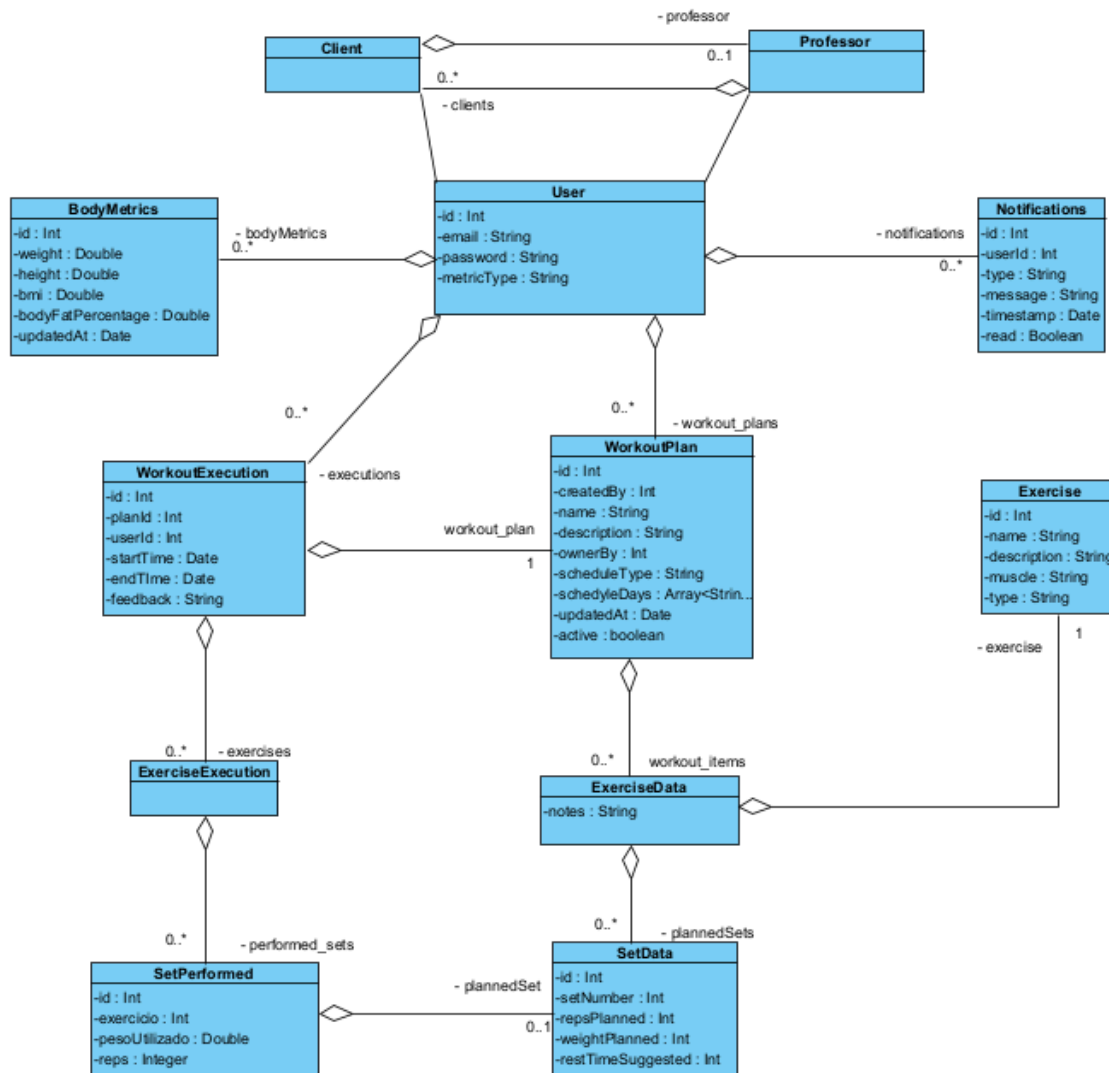


Figura 3: Diagrama de classes *PIM*

Com base no diagrama desenvolvido, vamos apresentar as diferentes funcionalidades do sistema suportadas por cada classe:

- **User:** Esta classe irá armazenar os dados dos utilizadores, com utilização principal para autenticação.
- **Cliente/Professor:** um cliente poderá ter um professor associado, e um professor terá uma lista de alunos associada.
- **BodyMetrics:** Esta classe representa as métricas corporais do utilizador. Serão guardadas várias instâncias para ser possível verificar o progresso dos utilizadores.

- **Notification:** Representa uma notificação enviada para o utilizador, que serão enviadas com base em certos eventos.
- **WorkoutPlan:** Esta classe representa um plano de treino, contendo informações gerais como nome e descrição, assim como o tipo de horário que o treino possui. Também possui os campos *createdBy* para permitir identificar quem criou o plano e *ownerId* para identificar a quem o plano pertence, de forma a podermos fazer a distinção dos planos que um professor criou para o seu cliente.
- **Exercise:** Esta classe vai guardar os dados relacionados a um exercício. Com isto, podemos montar uma biblioteca com todos os exercícios, a partir da qual os utilizadores selecionam um exercício para adicionar ao seu plano de treino.
- **ExerciseData:** Classe com o objetivo de representar um exercício específico dentro de um plano de treino. Serve como uma entidade de junção que relaciona um exercício com um plano de treino, armazenando informações específicas como, por exemplo, uma nota que pode ser deixada por quem criou o treino.
- **SetData:** Classe que representa um *set* planeado para um exercício no plano de treino. Guarda informações de repetições planeadas, peso, tempo de descanso e o número do *set*.
- **WorkoutExecution:** Representa a execução em concreto de um plano de treino por um utilizador. Regista quando um treino é iniciado, concluído e o *feedback* do utilizador para aquele treino. Também torna possível obter o histórico de treinos.
- **ExerciseExecution:** Possui o mesmo objetivo da classe **ExerciseData** mas para as entidades de execução.
- **SetPerformed:** Serve para registar os detalhes de um *set* realizado durante a execução de um treino. Está ligada a uma instância de *SetData* para o utilizador poder visualizar o que está planeado e registar o que executou.

2.5) Protótipos

Vamos agora apresentar os protótipos desenvolvidos para a interface da nossa aplicação. Para esta fase utilizamos a ferramenta *Figma* para nos auxiliar no processo.

2.5.1) Home page

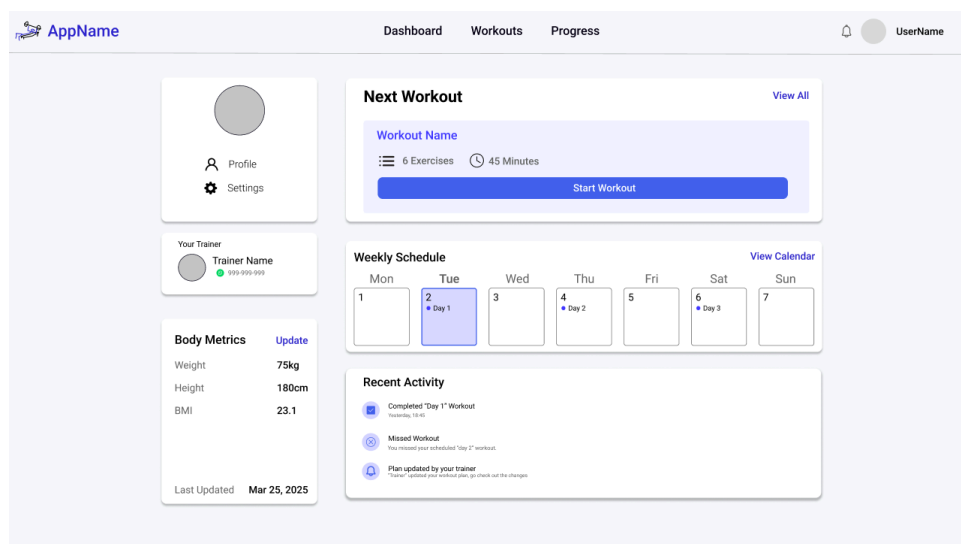
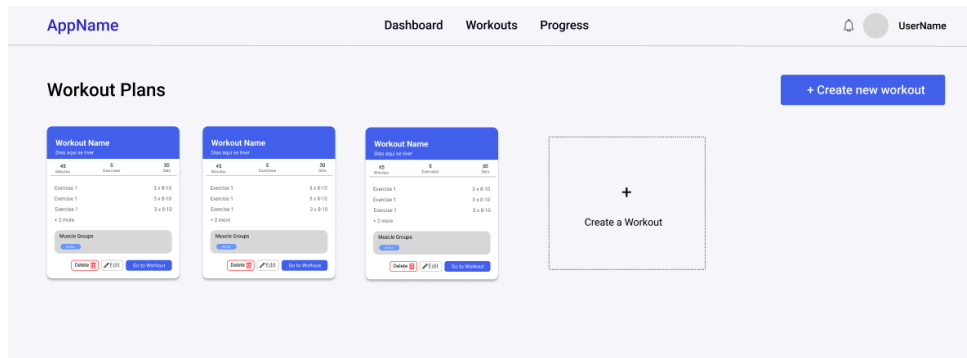
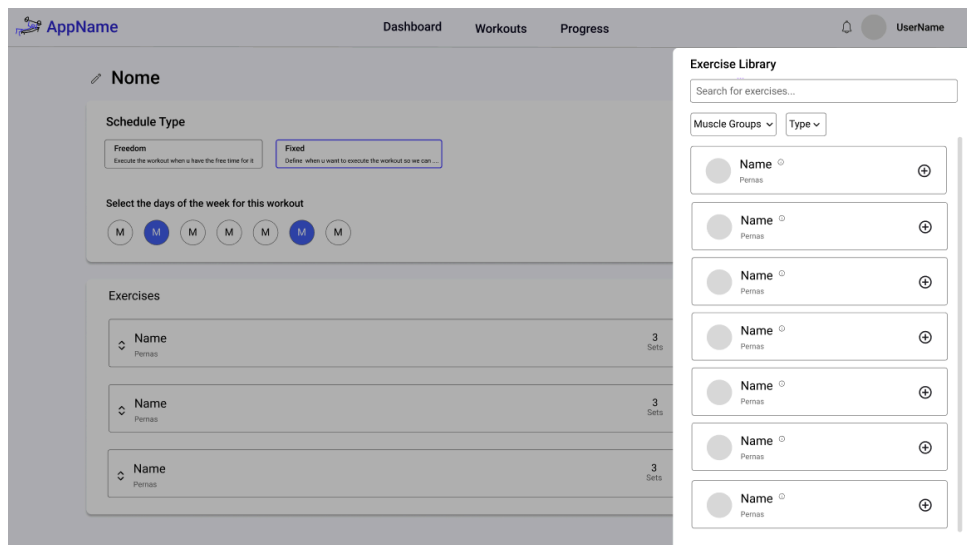
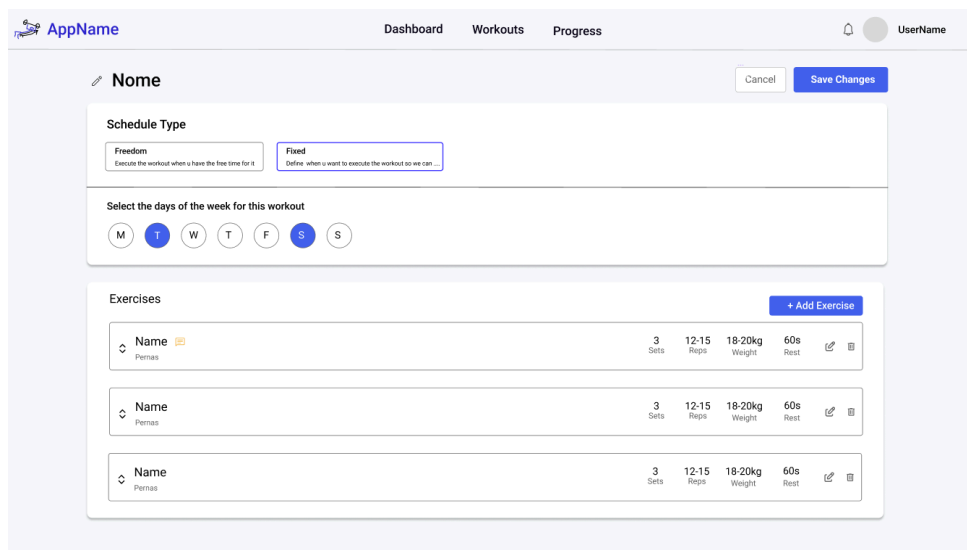


Figura 4: Protótipo da *home page*

2.5.2) Workouts List



2.5.3) Edit Workout



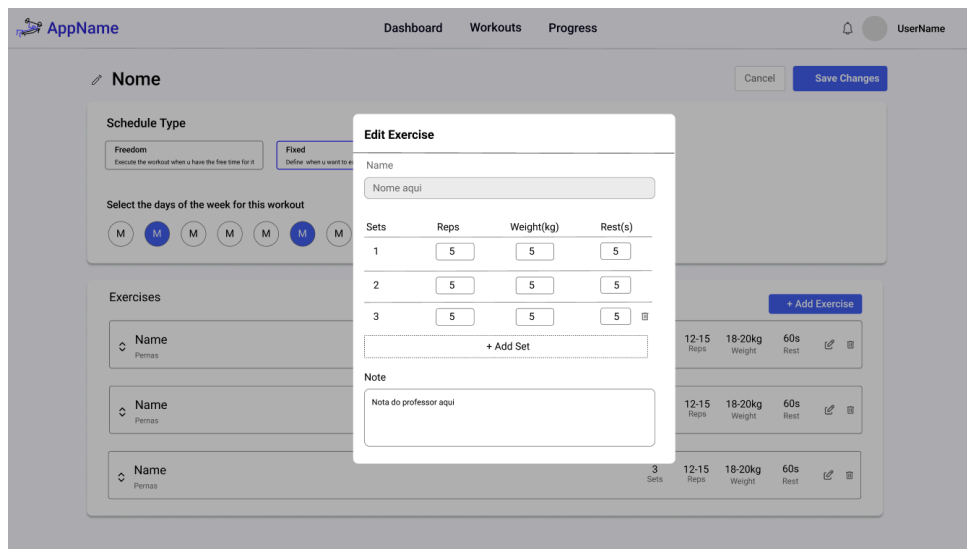


Figura 8: Protótipo da página de edição de planos de treino

2.5.4) Workout Execution

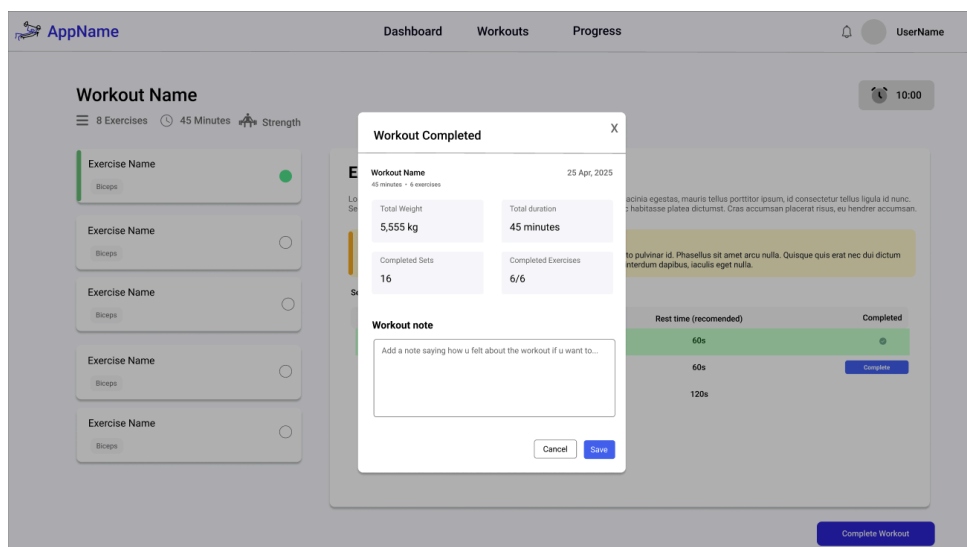
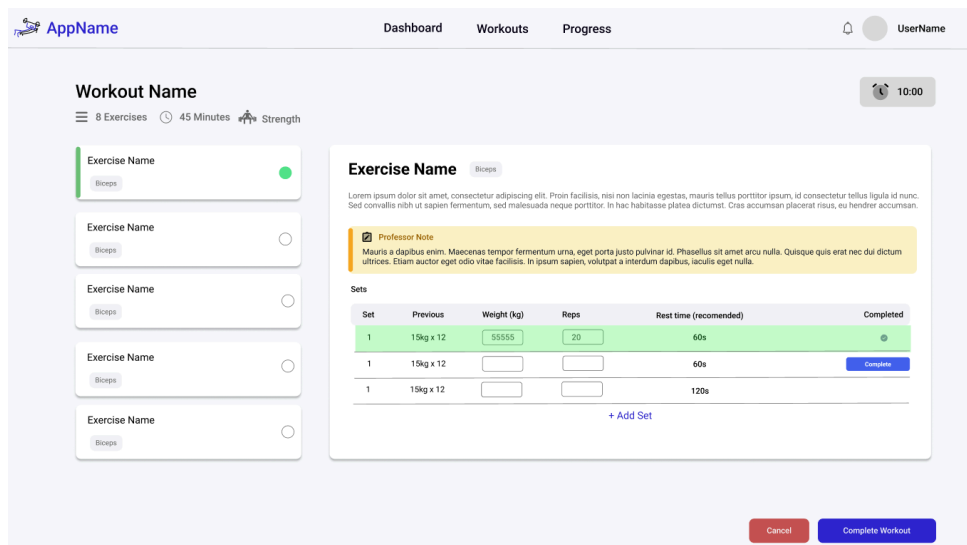


Figura 10: Protótipo da página de execução de um plano de treino

2.5.5) Progress Page

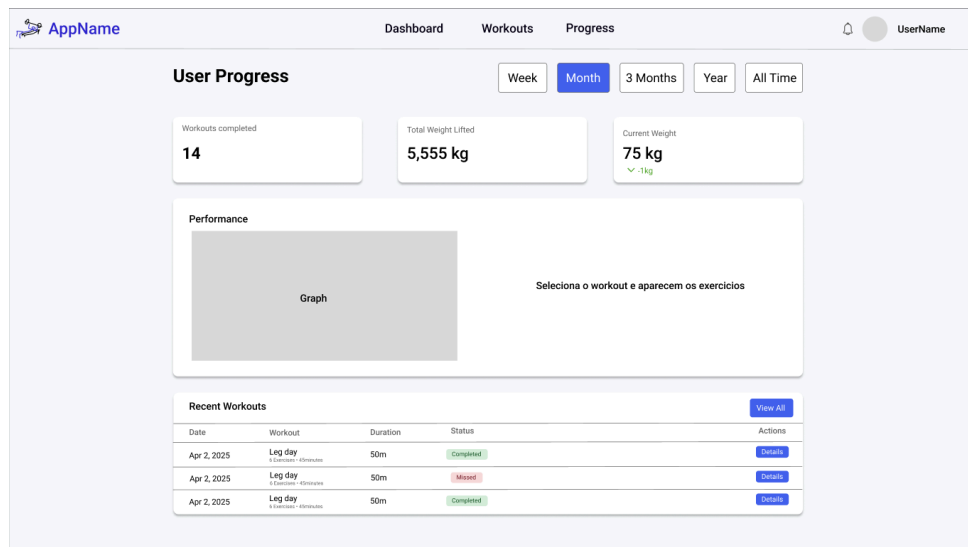


Figura 11: Protótipo da página de progresso de um utilizador

2.5.6) Clients Page

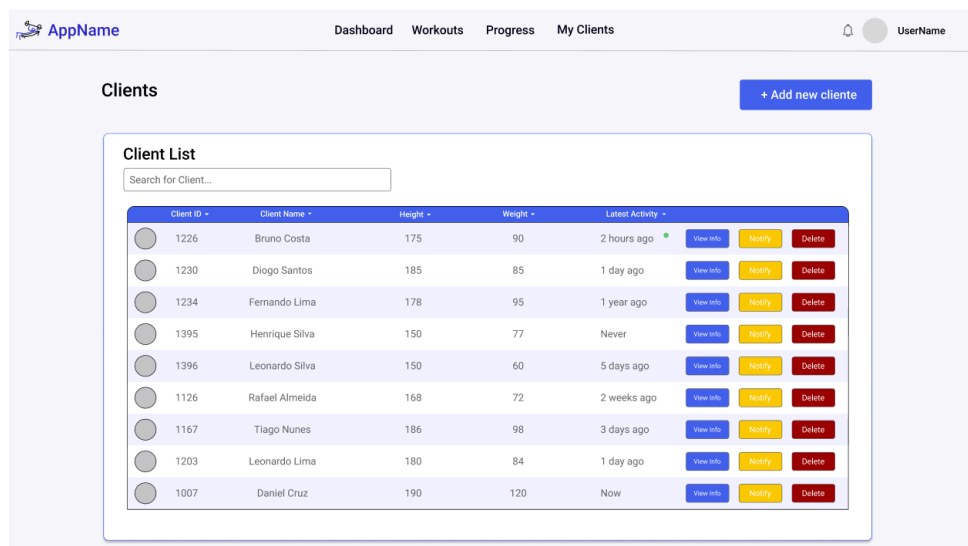


Figura 12: Protótipo da página com a lista de clientes de um professor

3) Implementação

Iremos agora abordar os tópicos relacionados com a implementação do sistema, incluindo a escolha das tecnologias, arquitetura do sistema e descrição das componentes do sistema.

3.1) Padrão arquitetural e tecnologias utilizadas

A arquitetura de um sistema de software é uma das decisões mais importantes para o processo de desenvolvimento, pois impacta diretamente a qualidade, o custo, o tempo de entrega e a sustentabilidade do produto final.

Para este sistema decidimos utilizar uma arquitetura aplicacional de três camadas (apresentação, lógica de negócio e dados). Cada camada tem a sua responsabilidade clara e bem definida, o que se traduz em benefícios para o desenvolvimento e manutenção da aplicação.

A camada de apresentação é a responsável pela interface da aplicação com que o utilizador interage. Para esta camada decidimos utilizar **Vue.js**, uma *framework* de **JavaScript** para o desenvolvimento *frontend*. Tem a vantagem de possuir reatividade, o que permite alterar elementos no *frontend* como respostas a eventos, sem ter que fazer o *reload* da página, promove também uma construção por componentes que facilita o desenvolvimento e manutenção da aplicação.



A camada de lógica de negócio processa os dados entre as camadas de apresentação e de dados. Para esta escolhemos a *framework* **Spring Boot**, uma *framework* utilizada para o desenvolvimento de aplicações em *Java*, responsável pelo controlo de dependências e gestão dos *controllers* que permitem a criação de *endpoints*.

A camada de dados trata do armazenamento e gestão de dados, interagindo com a base de dados para armazenar, obter e atualizar dados conforme lhe é requisitado. Para a base de dados, utilizamos *MySQL* que é um sistema de gestão de base de dados relacional, o qual consideramos ser o mais adequado ao nosso projeto. A ferramenta *ORM* escolhida foi o *Hibernate*, que é bastante compatível com **Spring** e facilita a implementação de operações de persistência, permitindo mapear objetos *Java* para tabelas relacionais de forma eficiente e simplificada.



3.2) Camada de Dados

A camada de dados da aplicação é modelada através das seguintes entidades e respetivos atributos, refletindo a estrutura de um sistema de gestão de fitness. Cada tabela representa uma entidade com um conjunto de atributos (colunas) e relações com outras entidades (chaves estrangeiras).

3.2.1) Entidades da Camada de Dados

3.2.1.1) Entidade: Aluno

Representa um aluno no sistema.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único do aluno.	int	PK, FK para User.id, NOT NULL
professor_id	Identificador do professor associado a este aluno.	int	FK para Professor.id, DEFAULT NULL

3.2.1.2) Entidade: BodyMetrics

Regista métricas corporais de um utilizador.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único da métrica corporal.	int	PK, AUTO_INCREMENT, NOT NULL
user_id	Identificador do utilizador a quem a métrica pertence.	int	FK para User.id, DEFAULT NULL
height	Altura do utilizador.	float	DEFAULT NULL
weight	Peso do utilizador.	float	DEFAULT NULL
bmi	Índice de Massa Corporal.	float	DEFAULT NULL
bodyFatPercentage	Porcentagem de gordura corporal.	float	DEFAULT NULL
updatedAt	Data e hora da última atualização da métrica.	datetime(6)	NOT NULL
metricType	Sistema de unidades da métrica.	enum	'IMPERIAL', 'METRIC', DEFAULT NULL

3.2.1.3) Entidade: Exercise

Representa um exercício físico individual.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único do exercício.	int	PK, AUTO_INCREMENT, NOT NULL
name	Nome do exercício.	varchar(255)	DEFAULT NULL
description	Descrição do exercício.	varchar(255)	DEFAULT NULL
type	Tipo de exercício.	varchar(255)	DEFAULT NULL
muscleGroup	Grupo muscular alvo do exercício.	varchar(255)	DEFAULT NULL
videoURL	URL de um vídeo com o exercício	varchar(255)	DEFAULT NULL
imageUrl	Imagem do exercício	varchar(255)	DEFAULT NULL

3.2.1.4) Entidade: ExerciseData

Associa exercícios a planos de treino, contendo dados para um exercício que se encontra dentro de um plano.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único dos dados do exercício.	int	PK, AUTO_INCREMENT, NOT NULL
workoutPlan_id	Identificador do plano de treino associado.	int	FK para WorkoutPlan.id, DEFAULT NULL
exercise_id	Identificador do exercício associado.	int	FK para Exercise.id, DEFAULT NULL
note	Notas adicionais sobre o exercício no plano.	varchar(255)	DEFAULT NULL
isDeleted	Indica se o registo foi marcado para eliminação.	bit(1)	NOT NULL

3.2.1.5) Entidade: ExerciseDataSet

Detalhes de um conjunto (*set*) de um exercício planeado (dentro do ExerciseData).

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único do conjunto de dados do exercício.	int	PK, AUTO_INCREMENT, NOT NULL
exerciseData_id	Identificador dos dados do exercício a que pertence.	int	FK para ExerciseData.id, DEFAULT NULL
setNumber	Número do conjunto.	int	DEFAULT NULL
weightPlanned	Peso planeado para este conjunto.	float	DEFAULT NULL
repsPlanned	Repetições planeadas para este conjunto.	int	DEFAULT NULL
restTimeSuggested	Tempo de descanso sugerido (float).	float	DEFAULT NULL
restTimeSuggestion	Tempo de descanso sugerido (decimal).	decimal(21,0)	DEFAULT NULL
restTimeSugested	Tempo de descanso sugerido (float, typo).	float	DEFAULT NULL
isDeleted	Indica se o registo foi marcado para eliminação.	bit(1)	NOT NULL

3.2.1.6) Entidade: ExerciseExecution

Regista a execução de um exercício específico dentro de uma execução do plano de treino.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único da execução do exercício.	int	PK, AUTO_INCREMENT, NOT NULL
exerciseData_id	Identificador dos dados do exercício original.	int	FK para ExerciseData.id, DEFAULT NULL
workoutExecution_id	Identificador da execução do plano de treino a que pertence.	int	FK para WorkoutExecution.id, DEFAULT NULL
status	Estado da execução do exercício.	varchar(50)	DEFAULT NULL
startTime	Data e hora de início da execução.	timestamp	DEFAULT NULL
endTime	Data e hora de fim da execução.	timestamp	DEFAULT NULL

3.2.1.7) Entidade: ExerciseExecutionSet

Detalhes de um conjunto (*set*) de um exercício **executado**.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único do conjunto de execução do exercício.	int	PK, AUTO_INCREMENT, NOT NULL
exerciseExecution_id	Identificador da execução do exercício a que pertence.	int	FK para ExerciseExecution.id, DEFAULT NULL
setNumber	Número do conjunto executado.	int	DEFAULT NULL
repsPerformed	Repetições realizadas neste conjunto.	int	DEFAULT NULL
weightPerformed	Peso utilizado neste conjunto.	float	DEFAULT NULL
restTimePerformed	Tempo de descanso realizado (float).	float	DEFAULT NULL
resTimePerformed	Tempo de descanso realizado (decimal, tipo).	decimal(21,0)	DEFAULT NULL
set_data_id	Identificador do conjunto de dados do exercício planeado.	int	FK para ExerciseDataSet.id, DEFAULT NULL
isCompleted	Indica se o conjunto foi completado.	bit(1)	NOT NULL

3.2.1.8) Entidade: Notification

Regista e gere notificações para os utilizadores.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único da notificação.	bigint	PK, AUTO_INCREMENT, NOT NULL
title	Título da notificação.	varchar(255)	DEFAULT NULL
message	Corpo da mensagem da notificação.	varchar(255)	DEFAULT NULL
priority	Prioridade da notificação.	int	DEFAULT NULL
createdAt	Data e hora de criação da notificação.	datetime(6)	NOT NULL
read	Indica se a notificação foi lida (tinyint).	tinyint(1)	DEFAULT NULL
receiver_id	Identificador do recetor da notificação.	int	FK para User.id, DEFAULT NULL
type	Tipo de notificação.	enum	'WORKOUT_CREATED', 'WORKOUT_UPDATE', 'PROFESSOR_ASSIGNED', 'PROFESSOR_UNASSIGNED', 'PROFESSOR_NOTIFY', 'SCHEDULED_WORKOUT', 'WORKOUT_FINISHED', NOT NULL
receiver	Identificador do recetor da notificação (redundante com receiver_id).	int	FK para User.id, NOT NULL
is_read	Indica se a notificação foi lida (bit).	bit(1)	DEFAULT NULL

3.2.1.9) Entidade: Professor

Representa um professor no sistema.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único do professor.	int	PK, FK para User.id, NOT NULL

3.2.1.10) Entidade: User

Representa um utilizador genérico no sistema.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único do utilizador.	int	PK, AUTO_INCREMENT, NOT NULL
name	Nome completo do utilizador.	varchar(255)	DEFAULT NULL
email	Endereço de email do utilizador.	varchar(255)	UNIQUE, DEFAULT NULL
password	Palavra-passe do utilizador (hash).	varchar(255)	DEFAULT NULL
metricType	Tipo de métrica preferida pelo utilizador.	varchar(255)	DEFAULT NULL

3.2.1.11) Entidade: WorkoutExecution

Regista a execução de um plano de treino por um utilizador.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único da execução do treino.	int	PK, AUTO_INCREMENT, NOT NULL
executionDate	Data da execução do treino.	date	DEFAULT NULL
workout_id	Identificador do plano de treino (redundante com workout_plan_id).	int	FK para WorkoutPlan.id, DEFAULT NULL
status	Estado da execução do treino.	varchar(20)	NOT NULL
feedback	Feedback textual da execução do treino (TEXT).	text	DEFAULT NULL
startTime	Data e hora de início da execução.	timestamp	DEFAULT NULL
endTime	Data e hora de fim da execução.	timestamp	DEFAULT NULL
feedback	Feedback textual da execução do treino (VARCHAR, typo).	varchar(255)	DEFAULT NULL
user_id	Identificador do utilizador que executou o treino.	int	FK para User.id, NOT NULL
workout_plan_id	Identificador do plano de treino executado.	int	FK para WorkoutPlan.id, NOT NULL

3.2.1.12) Entidade: WorkoutPlan

Define um plano de treino que pode ser atribuído a utilizadores.

Atributo	Descrição	Tipo	Chave/Observação
id	Identificador único do plano de treino.	int	PK, AUTO_INCREMENT, NOT NULL
name	Nome do plano de treino.	varchar(255)	DEFAULT NULL
createdBy	Identificador do utilizador que criou o plano.	int	FK para User.id, DEFAULT NULL
owner_id	Identificador do utilizador que é o proprietário do plano.	int	FK para User.id, DEFAULT NULL
scheduleType	Tipo de agendamento do plano.	enum	'Fixed', 'Free', DEFAULT NULL
scheduleDays	Dias da semana programados para o plano (texto livre).	text	DEFAULT NULL
description	Descrição detalhada do plano de treino.	varchar(255)	DEFAULT NULL
updatedAt	Última data de atualização do plano.	date	DEFAULT NULL
active	Indica se o plano está ativo.	bit(1)	NOT NULL
is_deleted	Indica se o plano foi marcado para eliminação.	bit(1)	DEFAULT NULL

3.2.1.13) Entidade: workout_plan_scheduled_days

Tabela de junção para especificar os dias agendados de um plano de treino.

Atributo	Descrição	Tipo	Chave/Observação
workout_plan_id	Identificador do plano de treino.	int	FK para WorkoutPlan.id, NOT NULL
scheduledDays	Dia da semana programado para o treino.	enum	'FRIDAY', 'MONDAY', 'SATURDAY', 'SUNDAY', 'THURSDAY', 'TUESDAY', 'WEDNESDAY', DEFAULT NULL

3.2.2) Relações entre Entidades

As relações entre as entidades são estabelecidas através de chaves estrangeiras, permitindo a integridade e a conectividade dos dados:

- A entidade **User** serve como base para utilizadores genéricos, sendo **Aluno** e **Professor** especializações da mesma (relação 1:1, onde `Aluno.id` e `Professor.id` referenciam `User.id`).
- Um **Professor** pode supervisionar múltiplos **Aluno(s)** (relação 1:N entre **Professor** e **Aluno** através de `Aluno.professor_id`).
- Métricas corporais (**BodyMetrics**) são registadas para um **User** (relação 1:N entre **User** e **BodyMetrics** através de `BodyMetrics.user_id`).
- Um **WorkoutPlan** é criado e pode ser possuído por um **User** (através dos atributos `createdBy` e `owner_id`).
- Um **WorkoutPlan** é composto por **ExerciseData**, que por sua vez referencia **Exercise**. Isso estabelece uma relação N:M entre **WorkoutPlan** e **Exercise** através de **ExerciseData**.
- Cada **ExerciseData** pode ter múltiplos **ExerciseDataSet(s)** para detalhar os conjuntos planeados (relação 1:N).
- As execuções de exercícios (**ExerciseExecution**) registam a realização de **ExerciseData** e estão associadas a uma execução de plano de treino (**WorkoutExecution**).
- Cada **ExerciseExecution** é detalhada por **ExerciseExecutionSet(s)**, que também referenciam **ExerciseDataSet(s)** (relação 1:N e N:M implícita).
- As notificações (**Notification**) são enviadas para um **User** (relação 1:N, através de `receiver_id` e `receiver`).
- Um **WorkoutExecution** regista a execução de um **WorkoutPlan** por um **User** (relações 1:N).
- Os dias agendados para um **WorkoutPlan** são especificados na tabela `workout_plan_scheduled_days` (relação 1:N).

Esta estrutura de dados detalhada fornece uma base robusta para a implementação e gestão das funcionalidades da aplicação de fitness, garantindo a integridade e a consistência da informação.

3.3) Camada da lógica de negócio

Como mencionado anteriormente, o servidor aplicacional responsável pela lógica de negócio será um servidor *Java* a utilizar a *framework Spring*. Esta divide a implementação em diferentes módulos, que são *models*, *services*, e *controllers*.

- **Models** contêm as classes que irão ser mapeadas para as entidades persistidas na base de dados.
- **Services** são classes que possuem os métodos da lógica de negócio.
- **Controllers** são responsáveis pela gestão da receção de pedidos e pelas respostas a esses pedidos. Os *controllers* usam os *services* para executar as funcionalidades da aplicação e construir as respostas aos pedidos a devolver.

3.3.1) Modelos

Com base nestes detalhes, foi desenvolvido o diagrama de classes **PSM (Platform Specific Model)**, relativo as entidades persistidas na base de dados.

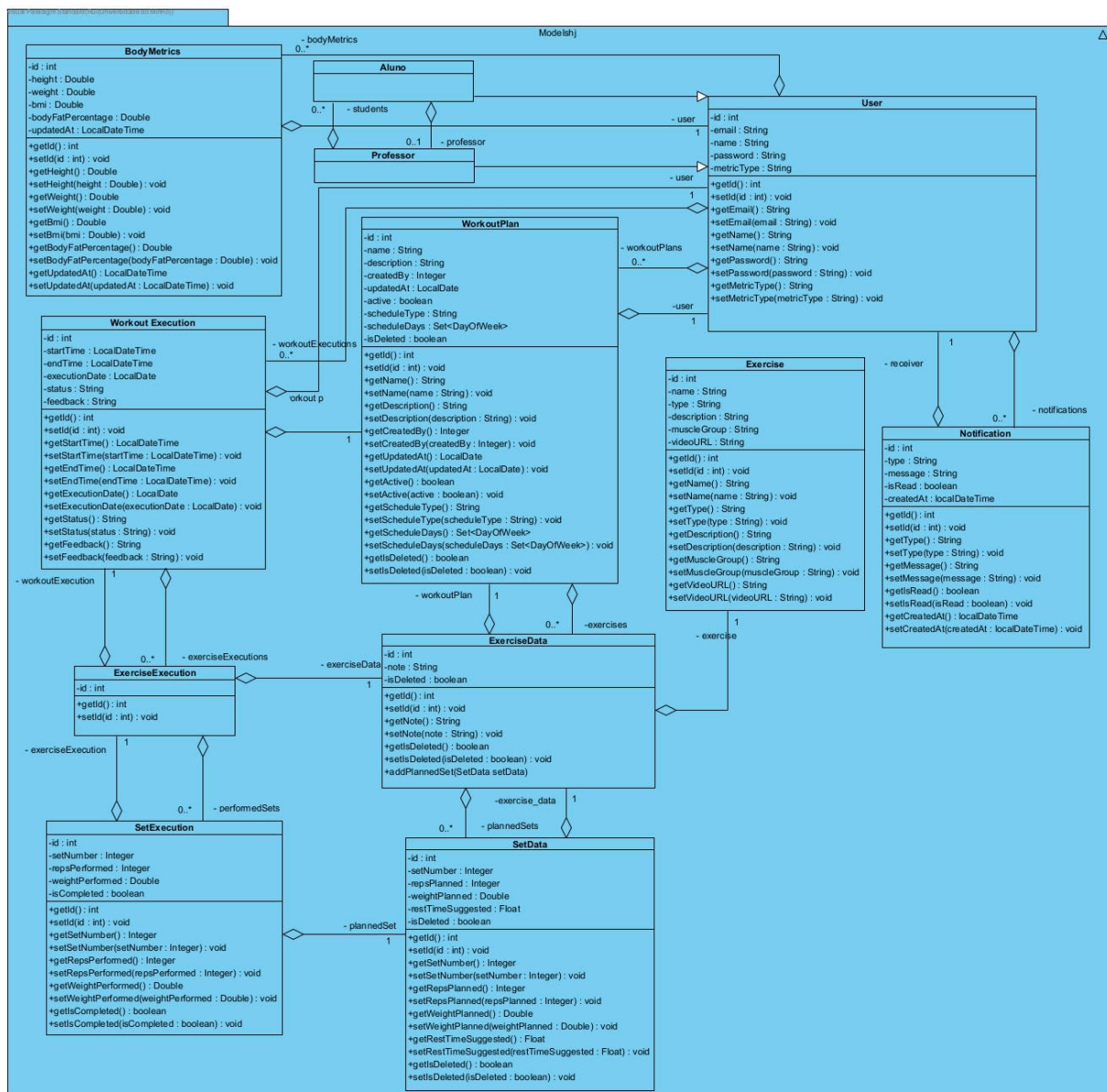


Figura 17: Diagrama de classes *PSM* para Modelos

3.3.2) Serviços

Vamos agora apresentar o diagrama de classes *PSM* relativo aos serviços desenvolvidos que implementam as funcionalidades da lógica de negócio.

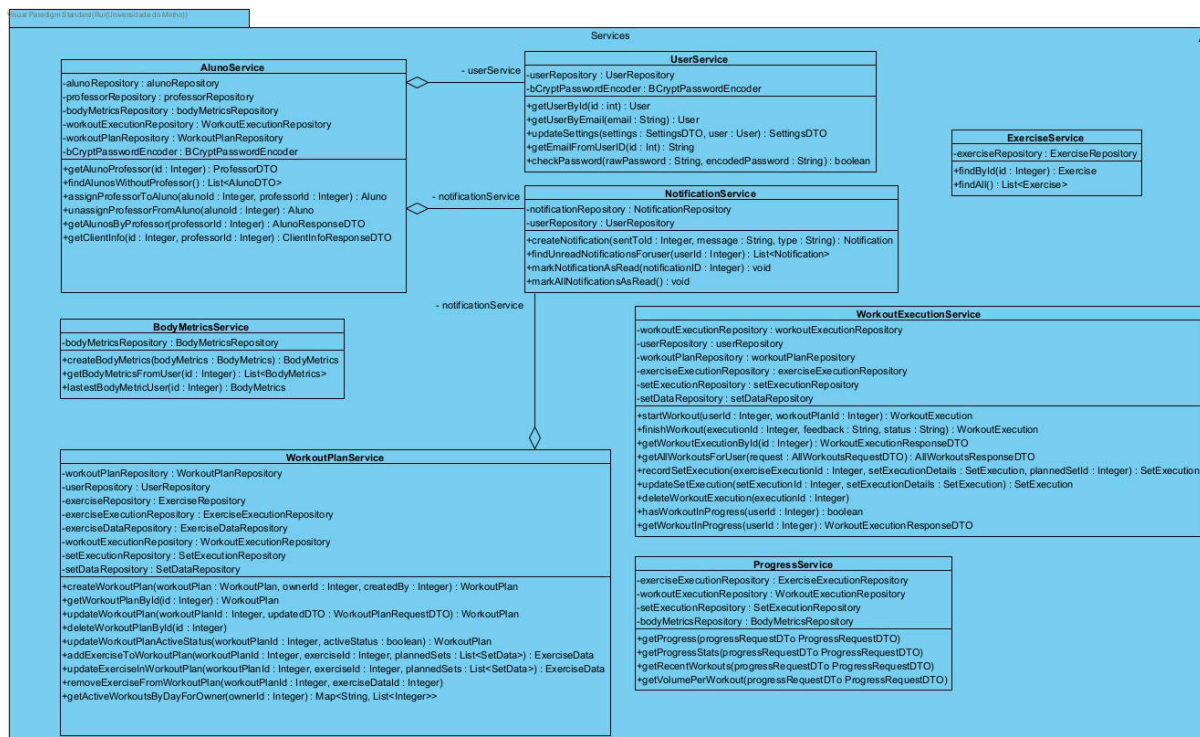


Figura 18: Diagrama de classes *PSM* para Serviços

- **UserService** - responsável por gerir os utilizadores. É através dele que é gerido o registo de novas contas na aplicação e se verificam as credenciais no *login*.
- **AlunoService** - responsável pelas operações relacionadas aos alunos. Lida principalmente com ações relacionadas à relação aluno-professor.
- **BodyMetricsService** - gere as operações relacionadas às *bodyMetrics* dos utilizadores.
- **NotificationExercise** - responsável pela lógica relacionada a notificações. É utilizado por outros serviços para a criação de notificações perante um certo evento.
- **WorkoutPlanService** - responsável pelas operações relacionadas ao planeamento de um plano de treino.
- **WorkoutExecutionService** - responsável pelas operações relacionadas à execução de um plano de treino.
- **ProgressService** - este serviço é responsável pela lógica relacionada ao progresso de um utilizador.

Existem também os serviços *JWTService* e *AuthService*. O *JWTService* é responsável pelas operações de geração e validação de tokens utilizadas na autenticação dos utilizadores. Este é utilizado como auxiliar ao *AuthService*, que contém todas as funcionalidades de autenticação, como registo, login e geração e validação de tokens.

3.3.3) *Controllers*

Vamos agora apresentar os *controllers* e os *endpoints* desenvolvidos para cada um. Estes apresentam a lógica de resposta aos pedidos feitos para os *endpoints* e utilizam as funcionalidades dos *services*.

AuthController

- **POST: /register** - registo de alunos.
- **POST: /login** - login de utilizadores.
- **POST: /logout** - logout de utilizadores.

UserController

- **GET: /dashboard-workout** - responsável por enviar os dados necessários para o *display* do plano de treino na página principal, podendo não existir, estar a decorrer um treino, ou devolver o próximo treino agendado.
- **POST: /settings** - atualizar as definições de um utilizador.

AlunoController

- **GET: /** - devolve todos os alunos sem professor.
- **GET: /professor** - devolve o professor de um aluno.
- **GET: /{id}** - devolve todos os alunos de um professor.
- **POST: /{alunoId}/assign/{professorId}** - associa um aluno a um professor.
- **DELETE: /{alunoId}/unassign** - remove a associação aluno-professor.
- **GET: /info/{id}** - devolve a informação de um cliente para ser exibida quando o seu professor pretende ver os detalhes do cliente.

BodyMetricsController

- **POST: /** - para criar um objeto de *BodyMetrics*.
- **GET: /user/{userId}** - obter a lista de objetos *BodyMetrics* de um utilizador.
- **GET: /{userId}/latest** - obter as *bodyMetrics* mais recentes do utilizador.

ExerciseController

- **GET: /** - obter a biblioteca de exercícios.
- **GET: /{id}** - um exercício em específico.

NotificationController

- **GET: /** - notificações não lidas.
- **POST: /** - criar uma notificação.
- **GET: /all** - todas as notificações.
- **PUT: /{id}** - marcar uma notificação como lida.
- **PUT: /markAll** - marca todas as notificações como lidas.

WorkoutPlanController

- **GET: /user/{ownerId}** - obter os planos de treino de um certo utilizador.
- **POST: /** - criar um plano de treino.
- **GET: /{id}** - obter um plano de treino pelo seu *id*.
- **DELETE: /{id}** - apagar um plano de treino pelo seu *id*.

- **PUT: /{id}** - atualizar um plano de treino.
- **PUT: /{id}/activate** - ativar um plano de treino.
- **PUT: /{id}/deactivate** - desativar um plano de treino.
- **POST: /{workoutPlanId}/exercises** - adicionar um exercício ao plano de treino.
- **PUT: /{workoutPlanId}/exercises/{id}** - atualizar um exercício no plano de treino.
- **DELETE: /{workoutPlanId}/exercises/{id}** - remover um exercício do plano de treino.
- **GET: /weekly-schedule/active** - obter os planos de treino que o utilizador tem planeados para a semana atual.

WorkoutExecutionController

- **GET: /in-progress/{userId}** - verificar se o utilizador tem um *workout* em progresso e devolver se tiver.
- **POST: /start** - iniciar a execução de um plano de treino.
- **PUT: /{executionId}/finish** - terminar a execução.
- **GET: /{id}** - obter uma execução de um plano de treino por *id*.
- **POST: /exercises/{exerciseExecutionId}/sets** - fazer o registo da execução de um *set*.
- **PUT: /sets/{setExecutionId}** - atualizar os dados de um *set*.
- **POST: /all** - obter todas as execuções de planos de treino.

ProgressController

- **GET: /dashboard** - obter todos os dados relativos ao progresso para a *dashboard*.
- **GET: /stats** - obter as estatísticas de progresso.
- **GET: /recent-workouts** - obter os 5 *workouts* completados mais recentes.

3.4) Camada de apresentação

Como referido anteriormente a camada de aplicação foi desenvolvida utilizando a *framework* **Vue**.

3.4.1) Interfaces desenvolvidas

Tendo como base os *mockups* desenvolvidos previamente, fizemos as seguintes interfaces.

3.4.1.1) Registo

Na página de registo, como é uma página de autenticação, não disponibiliza a barra de navegação. O utilizador deve inserir os dados necessários como nome, email e password, que deve ser confirmada ao ser escrita duas vezes e por fim submeter.

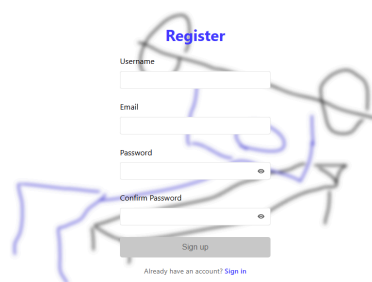


Figura 19: Página de *registo*

3.4.1.2) Login

Na página de *login*, como é também uma página de autenticação, não disponibiliza a barra de navegação. O utilizador deve então inserir o seu email e palavra-passe para realizar o *login*. No caso em que as credenciais inseridas estejam erradas ou não existam, é apresentado um erro adequado.



Figura 20: Página de *login*

3.4.1.3) Home

Depois de iniciar sessão o utilizador é redirecionado para a página principal. Esta página e as restantes já disponibilizam uma barra de navegação para o utilizador, que apresenta o logótipo da aplicação (que também funciona como atalho para a *home page*) e atalhos para as página de Planos de Treino, *Home Page* e página de progresso. No caso de o utilizador ser um professor também possui um atalho para a página com a lista dos seus clientes. Inclui também do lado direito um botão para visualizar as suas notificações e um atalho para realizar o logout.

Relativamente à página em si, do lado esquerdo o utilizador possui informações relativas a si, como os botões para poder alterar as suas definições ou efetuar o *logout* se pretender. Possui também uma secção que apenas aparece caso o utilizador tenha um professor, onde apresenta a informação do mesmo. Em baixo temos uma secção que demonstra a informação das medidas corporais do utilizador com a opção para alterar as mesmas. Do lado direito, como podemos visualizar na fig.19, o utilizador possui uma secção que lhe indica o próximo plano de treino que tem agendado. Esta secção tem três estados, o que está apresentado na imagem onde demonstra o próximo treino marcado. Caso exista um treino em execução irá mostrar o treino a decorrer, assim como um atalho para o mesmo e caso não existam planos marcados irá aparecer uma mensagem a indicar essa informação e um atalho que leva à página onde o utilizador pode explorar os seus planos (ou até criar um novo). Depois temos uma secção que mostra ao utilizador os planos agendados para a semana atual, onde é possível clicar nos nomes dos planos para ser redirecionado para a página de detalhes. Por fim temos a atividade recente que mostra ao utilizadores “acontecimentos” recentes, como, por exemplo, ter completado um treino.

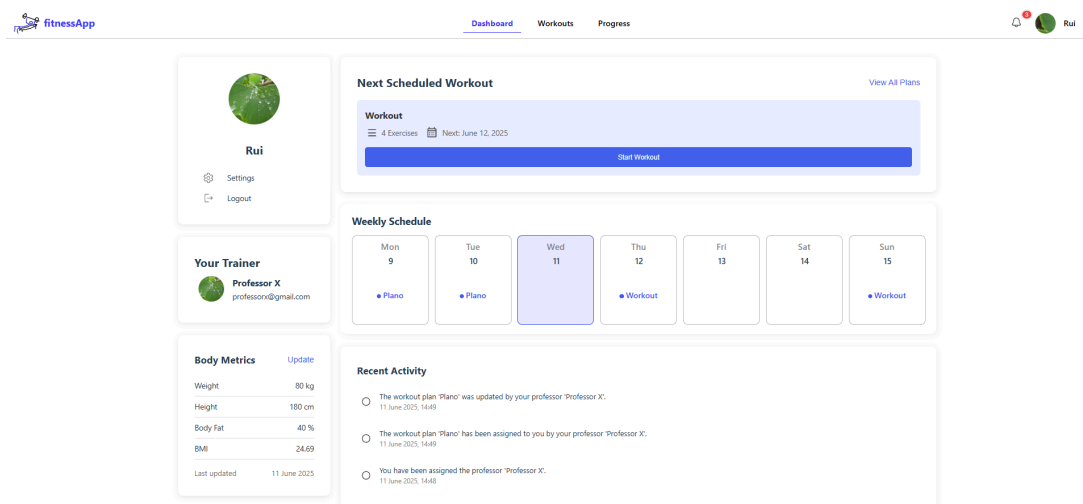


Figura 21: Página inicial

Figura 22: Modal *BodyMetrics*

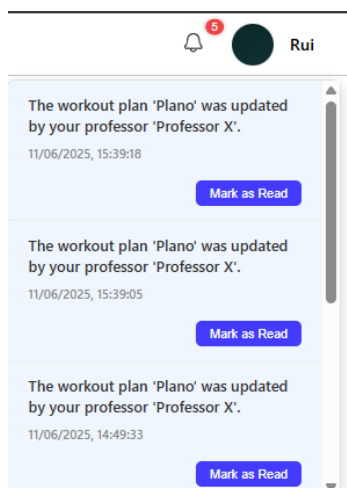


Figura 23: Modal Notificações

Figura 24: Modal Definições

3.4.1.4) Planos de treino

Esta página contém todos os planos de treino do utilizador. No canto superior direito existe um botão para criar um plano de treino novo, que abre a modal apresentada na fig.26. A página está dividida em duas secções, uma que demonstra os planos de treino inativos e outra que demonstra os planos de treino ativos, em que cada uma possui um ícone de informação para guiar o utilizador no propósito de ativar/desativar um plano de treino. Em cada secção são listados os planos, e cada plano possui informações sobre o seu nome, quem o criou, quantos e quais exercícios têm e o seu número de *sets*. Oferece também opções para cada plano como um botão para ser redirecionado para a página de edição, um botão para desativar/ativar o plano e um botão para ser redirecionado para a página de detalhes desse plano.

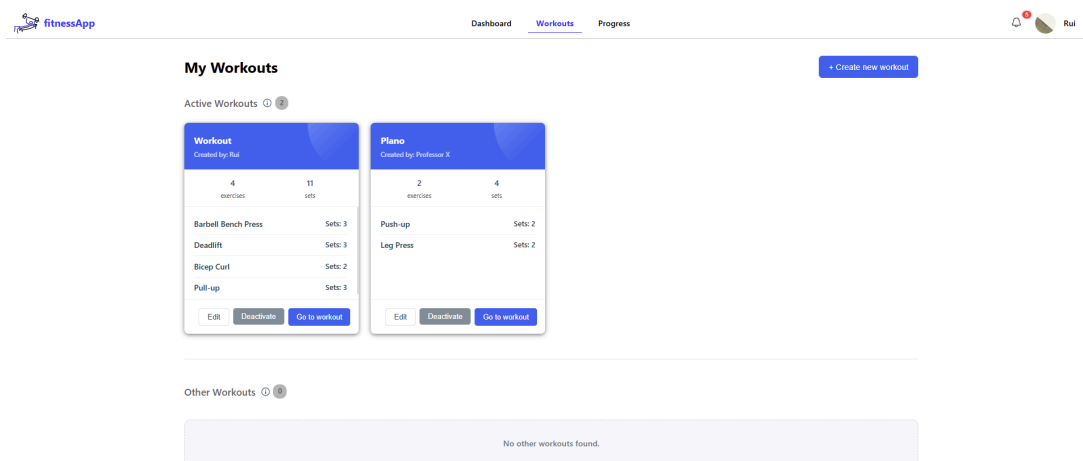


Figura 25: Página dos planos de treino

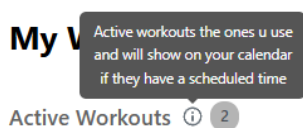


Figura 26: Informação sobre planos ativos

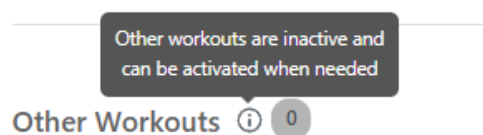


Figura 27: Informação sobre planos inativos

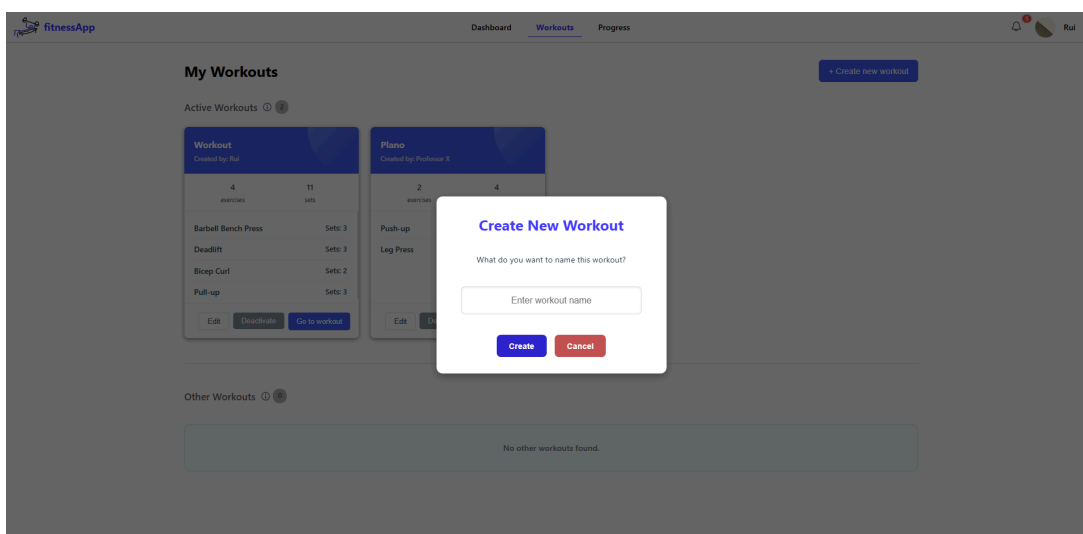


Figura 28: Modal de criação de planos de treino

3.4.1.5) Editar Plano de Treino

Esta é a página onde o utilizador realiza a edição dos seus planos de treino. O utilizador pode alterar o nome do plano ou a sua descrição através do ícone no seu lado esquerdo, e de seguida escolher o tipo de horário do plano. Caso escolha o horário fixo deverá seleccionar os dias em que pretende realizar o treino.

Depois tem a secção dos exercícios que tem uma opção para adicionar exercícios. Este botão abre a modal que possui a biblioteca de exercícios, onde o utilizador pode filtrar por grupo muscular ou tipo de exercício e seleccionar os exercícios que pretende adicionar. Após adicionar os exercícios o utilizador tem a opção de editar ou remover o exercício. Se seleccionar editar exercício irá abrir a modal

de edição do exercício onde pode inserir os *sets* que pretende com os dados relativos, incluindo uma nota adicional.

Plano

No description

Go Back Save Changes Delete Workout

Schedule Type

Freedom
Execute the workout when I have the free time for it

Fixed
Select the day/s of the week for this workout

Select the days of the week for this workout

M T W T F S S

Exercises

+ Add Exercise

	Sets	Reps	Weight	Rest	
Push-up Chest	2	6	6kg	6s	
Leg Press Legs	2	15	15kg	5-15s	

Figura 29: Página de edição de um plano de treino

Exercise Library

Search for exercises...

Muscle Group Type

Barbell Bench Press
Chest
BODYBUILDING

Barbell Squat
Legs
BODYBUILDING

Deadlift
Back
BODYBUILDING

Overhead Press
Shoulders
BODYBUILDING

Bicep Curl
Biceps
BODYBUILDING

Triceps Extension
Triceps
BODYBUILDING

Figura 30: Modal de biblioteca de exercícios

Edit Exercise

Exercise Name
Push-up

Set	Reps	Weight (kg)	Rest (s)
1	6	6	6
2	6	6	6

+ Add Set

Note
Add a note...

Cancel Save Changes

Figura 31: Modal de edição de um Set

3.4.1.6) Detalhes de um plano

Esta página exibe os detalhes de um plano de treino. No lado esquerdo, o utilizador pode seleccionar o exercício que pretende visualizar e são depois exibidos os seus detalhes como o nome, grupo muscular que treina, descrição e os *sets* planeados. Possui também a opção para voltar à página dos planos de treino ou iniciar o treino, que abre uma modal a pedir confirmação da ação.

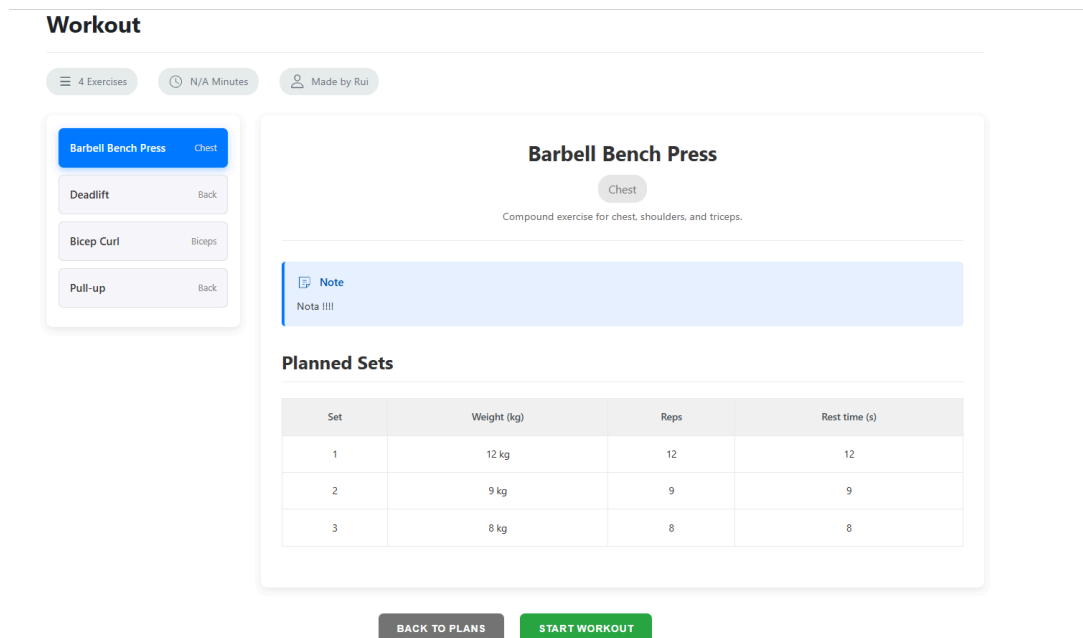


Figura 32: Página com detalhes de um plano

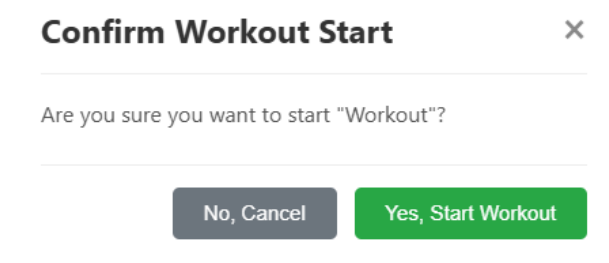


Figura 33: Modal para confirmar inicio de um plano

3.4.1.7) Execução de um plano

Quando um utilizador inicia um treino é redirecionado para esta página. Em cima, com base na data de início do treino, é apresentado um contador com o tempo que já decorreu. Depois, no lado esquerdo, são apresentados os exercícios a completar. Caso todos os *sets* tenham sido realizados, aparece um ícone *checkmark* a indicar que todos os *sets* foram realizados. Do lado direito são apresentados os detalhes do exercício e a informações relativas aos *sets* a realizar, assim como opção para acrescentar *sets* adicionais e registar os *sets* efetuados. O utilizador pode também editar um *set* caso tenha cometido algum erro no registo.

Workout

00:03:07

4 Exercises

Date: 2025-06-11

Barbell Bench Press

Chest

Deadlift

Back

Bicep Curl

Biceps

Pull-up

Back

Barbell Bench Press

Chest

Compound exercise for chest, shoulders, and triceps.

Sets

Set	Planned Weight x Reps	Performed Weight (kg)	Performed Reps	Rest time (s)	Actions
1	12kg x 12	6kg	6	12 s	<div>✓</div> <div></div>
2	9kg x 9	<div>Weight</div>	<div>Reps</div>	9 s	<div>Record Set</div>
3	8kg x 8	<div>Weight</div>	<div>Reps</div>	8 s	<div>Record Set</div>

+ Add Set

CANCEL WORKOUT

COMPLETE WORKOUT

Figura 34: Página de execução de um treino

Quando o utilizador termina e decide completar o treino (clicando no botão “*Complete Workout*”), é exibida uma modal com um resumo do treino e a opção para o utilizador fornecer um *feedback* acerca do mesmo.

Workout Completed

X

Workout Name

06/11/2025

5m 48s

36 kg

TOTAL VOLUME

1 / 11

SETS COMPLETED

0 / 4

EXERCISES DONE

Your Feedback / Notes

Any notes about this workout? Feeling, performance, etc.

SAVE WORKOUT

Figura 35: Modal para completar um treino

Depois do utilizador completar o treino, ele pode visualizar o treino executado, em que são demonstradas as mesmas informações mas é apenas disponibilizado um botão para voltar a página anterior. Para além disso, não permite inserir novos dados nos *sets* não completados.

3.4.1.8) Progresso

Esta é a página onde o utilizador e o seu professor podem visualizar o seu progresso. São disponibilizados botões para filtrar o progresso feito num certo período de tempo (Semana, Mês, Anual, Sempre).

A página pode ser dividida em três secções. Na primeira, o utilizador vê os detalhes sobre o número de treinos completados, volume de peso utilizado, e o progresso relativamente ao seu peso corporal nesse período de tempo. Caso tenha perdido peso, será demonstrado quanto peso perdeu com uma cor verde para indicar um bom progresso. Caso tenha ganho peso, será demonstrado com uma cor vermelha.

A seguinte secção possui um gráfico com o volume total utilizado por data, relativo a um plano de treino. Do lado direito, o utilizador pode seleccionar os planos de treino que tenham sido executados nesse período do tempo. Depois no gráfico serão exibidas as datas em que o treino foi realizado e o volume utilizado por treino, podendo o utilizador verificar se fez algum progresso.

Por fim, o utilizador tem uma lista dos últimos 5 treinos concluídos e dados sobre os mesmos, assim como uma opção para visualizar o treino completado. Possui também um botão que redireciona o utilizador para o seu histórico de execuções.

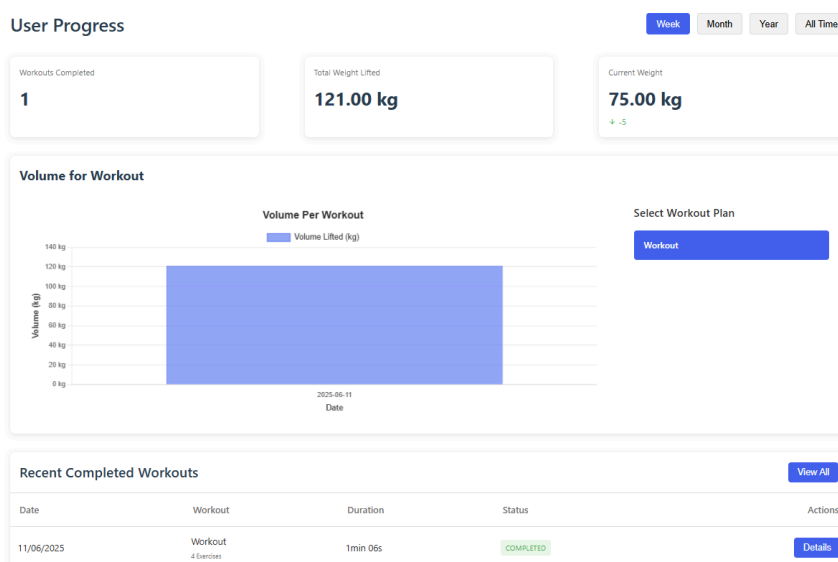
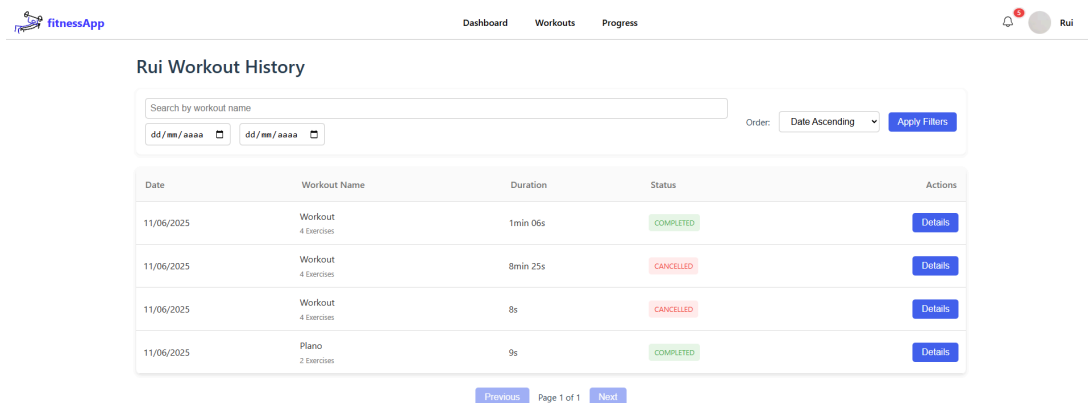


Figura 36: Página de progresso

3.4.1.9) Histórico de execuções

Esta página exibe o histórico de execuções de treinos, permitindo a filtragem por tempo, nome ou por data, assim como um botão para visualizar os detalhes da execução.



Rui Workout History

Search by workout name

dd/mm/yyyy dd/mm/yyyy

Order: Date Ascending Apply Filters

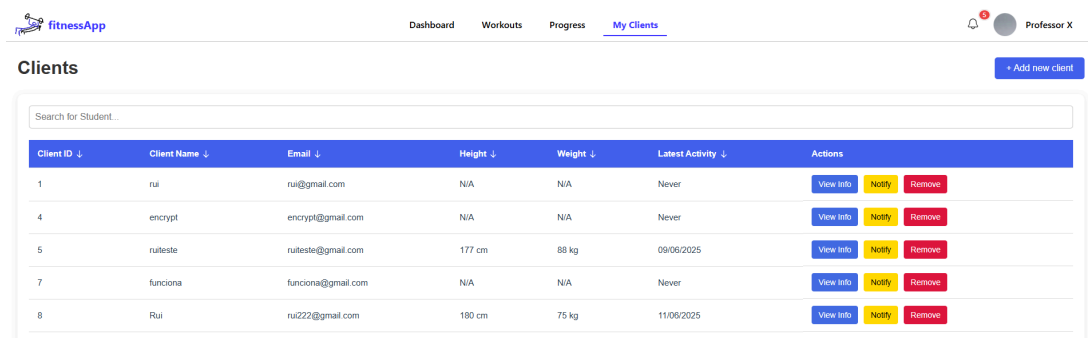
Date	Workout Name	Duration	Status	Actions
11/06/2025	Workout 4 Exercises	1min 06s	COMPLETED	Details
11/06/2025	Workout 4 Exercises	8min 25s	CANCELLED	Details
11/06/2025	Workout 4 Exercises	8s	CANCELLED	Details
11/06/2025	Plano 2 Exercises	9s	COMPLETED	Details

Previous Page 1 of 1 Next

Figura 37: Página do histórico de execuções

3.4.1.10) Clientes

Esta página é exclusiva a professores e exibe a lista dos seus clientes numa tabela com informações de cada cliente, assim como opções para ver a informação do cliente, enviar uma notificação ou removê-lo. Possui também um botão para adicionar um cliente, que abre a modal que podemos ver na fig.37.



fitnessApp

Dashboard Workouts Progress My Clients

Professor X

Client ID Client Name Email Height Weight Latest Activity Actions

1 rui rui@gmail.com N/A N/A Never View Info Notify Remove

4 encrypt encrypt@gmail.com N/A N/A Never View Info Notify Remove

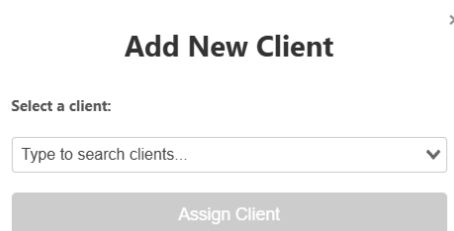
5 ruieste ruieste@gmail.com 177 cm 88 kg 09/06/2025 View Info Notify Remove

7 funciona funciona@gmail.com N/A N/A Never View Info Notify Remove

8 Rui ru22@gmail.com 180 cm 75 kg 11/06/2025 View Info Notify Remove

+ Add new client

Figura 38: Página com a lista de clientes



×

Add New Client

Select a client:

Type to search clients...

Assign Client

Figura 39: Modal para adicionar cliente

3.4.1.11) Informação de cliente

Quando o professor clica no botão “View Info” da lista de clientes, é redirecionado para esta página onde são exibidas as informações do cliente. A página possui informações básicas como o nome e email, as medidas corporais mais recentes e os planos que o professor criou para o cliente. É oferecida também ao professor a opção de visualizar o progresso do cliente ao clicar no botão “Check Client Progress”.

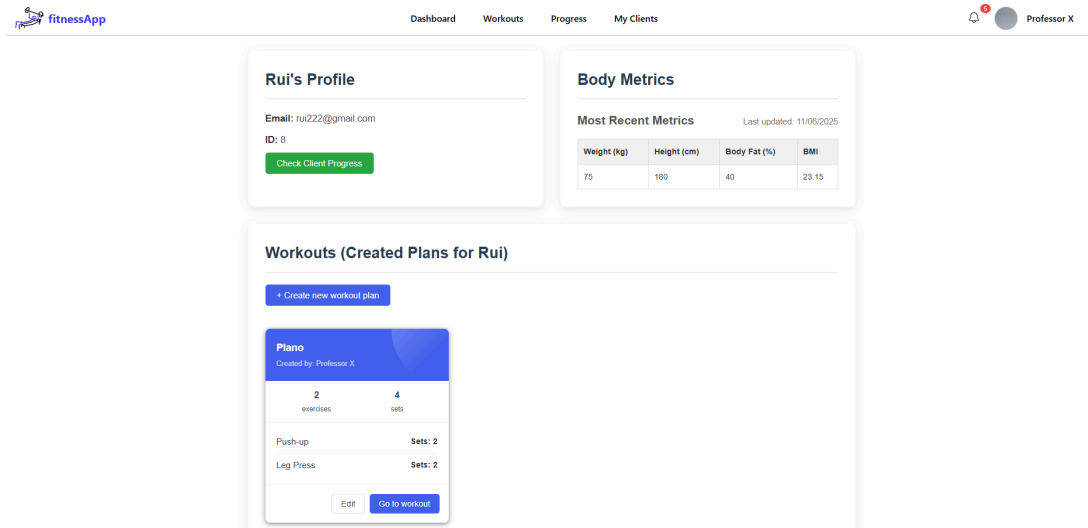


Figura 40: Página com informação sobre um cliente

3.4.2) Stores Pinia

Para a gestão do estado da sessão no desenvolvimento da *frontend* foi escolhido o **Pinia**, com o qual criamos as seguintes *stores*:

userStore.js

Esta pinia *store* gere e armazena os dados do utilizador. Utiliza *localStorage* para persistir os dados mesmo quando o *browser* é fechado.

```
const id = ref(localStorage.getItem('user_id') || null);
const name = ref(localStorage.getItem('name') || null);
const email = ref(localStorage.getItem('email') || null);
const metricType = ref(localStorage.getItem('metric_type') || 'METRIC');
const role = ref(localStorage.getItem('user_role') || null);
```

WorkoutStore.js

Esta *store* é responsável por armazenar e gerir os dados relativos aos planos de treino do utilizador e a biblioteca de exercícios.

```
const workoutExecution = ref(null);
const currentExerciseIndex = ref(0);
```


WorkoutExecutionStore.js Esta *store* é responsável por armazenar os dados relativos a um plano de treino que esteja em execução, o que permite que o utilizador navegue a aplicação sem perder os dados.

```
const workoutExecution = ref(null);  
const currentExerciseIndex = ref(0);
```

notificationStore.js

Esta *store* tem o objetivo de gerir as notificações recebidas pelo utilizador, possuindo também a lógica relativa às notificações. Desta forma o seu estado é persistido enquanto o utilizador navega a aplicação.

```
const notifications = ref([]);
```

4) Análise do sistema

4.1) Especificações e Capacidade de Escalabilidade da Aplicação

Utilizamos o **Google App Engine** com *auto-scaling*, onde o máximo de instâncias permitido é 20. Cada instância dispõe de 1 *core* e 0.6 GB de RAM, dedicados ao **frontend**. Dado que cada instância pode suportar até 250 utilizadores simultaneamente, o sistema, em teoria, consegue manter 5.000 utilizadores ativos ao mesmo tempo.

Para o **backend**, recorremos ao **Cloud Run**, também com *auto-scaling* ativado. Neste caso, o limite máximo é de 40 instâncias, cada uma equipada com 1 *core* e 512 MB de RAM. Cada instância é capaz de processar até 80 requisições simultâneas, o que significa que, em plena capacidade, podemos acomodar 3.200 utilizadores ao mesmo tempo.

No que toca à **base de dados**, utilizamos **Google Cloud SQL**, configurado com 1 *core*, 1.7 GB de RAM e 10 GB de armazenamento. Este componente revela-se o maior *bottleneck* do sistema, uma vez que optámos por uma configuração mais modesta para reduzir custos. Embora isso tenha limitado o benefício do *auto-scaling* no **frontend** e no **backend**, permitiu-nos prolongar o período de testes e implementação da aplicação.

4.2) Performance

Para testar a capacidade e performance do sistema desenvolvido, resolvemos utilizar o **Locust**, uma ferramenta *open-source* que permite realizar testes de carga, simulando operações de vários utilizadores simultaneamente. Com esta ferramenta foi possível replicar os comportamentos dos utilizadores na aplicação e avaliar a capacidade de resposta, robustez e escalabilidade do sistema.

Os testes que decidimos realizar englobam as principais funcionalidades que um utilizador, num cenário real, utilizaria : *login*, consulta dos planos de treino, criação de planos de treino e consulta do seu progresso.

Para isso, criamos um script *Locust* com 3 conjuntos de tarefas que permitem simular diferentes percursos a percorrer na aplicação:

1. *Login* e consulta de planos de treino

- **Login**: Simulação do processo de autenticação, através de um pedido *HTTP POST* com email e password, recebendo como resposta um *token* e o ID do utilizador.
- **Consulta da página de planos de treino**: Após a autenticação, o utilizador acede aos seus planos através de um pedido *HTTP GET* ao *endpoint* `/workout-plans/user/{userId}`. O *token* é incluído como *cookie* para garantir o acesso autorizado.

2. *Login* e consulta de progresso

- **Login**: Tal como no cenário anterior, o utilizador autentica-se com as suas credenciais.
- **Consulta do progresso**: O utilizador acede ao *dashboard* de progresso através de um pedido *HTTP POST* ao *endpoint* `/progress/dashboard`, incluindo o *userId* e o período de tempo desejado no pedido, bem como o *token* de autenticação nos *cookies*.

3. Login e criação de plano de treino

- **Login:** O utilizador autentica-se normalmente.
- **Criação do plano de treino:** É simulado um pedido *HTTP POST* ao *endpoint* /workout-plans com os dados mínimos necessários (nome e ID do utilizador) para criar um novo plano.

De seguida, utilizamos este *script* para realizar o *load testing*, onde será executado pelas várias *threads* que representam diferentes números de clientes em simultâneo (100, 500, 1500, 3000, 4000).

4.2.1) 100 threads

O primeiro teste de carga com 100 *threads* mostrou um desempenho estável por parte da aplicação. A linha verde no gráfico superior indica um crescimento consistente no número de requisições por segundo (RPS), sem quedas abruptas. Apesar de alguns picos pontuais nos tempos de resposta — especialmente no percentil 95, como se observa no gráfico inferior —, a aplicação conseguiu manter um tempo médio próximo dos 400ms, o que está dentro de uma margem aceitável para este nível de carga.

É importante notar que, mesmo com alguns aumentos temporários na latência, o sistema demonstrou boa capacidade de resposta, beneficiando do *auto-scaling* tanto no *Google App Engine* (*frontend*) como no *Cloud Run* (*backend*). Eventuais variações podem estar relacionadas à inicialização de novas instâncias ou ao tempo de resposta da base de dados, que foi dimensionada de forma mais conservadora com o objetivo de equilibrar desempenho e custos durante esta fase inicial de testes.

No geral, o sistema comportou-se bem com 100 *threads*, o que demonstra uma base sólida para cenários de maior carga, analisados nas próximas secções.

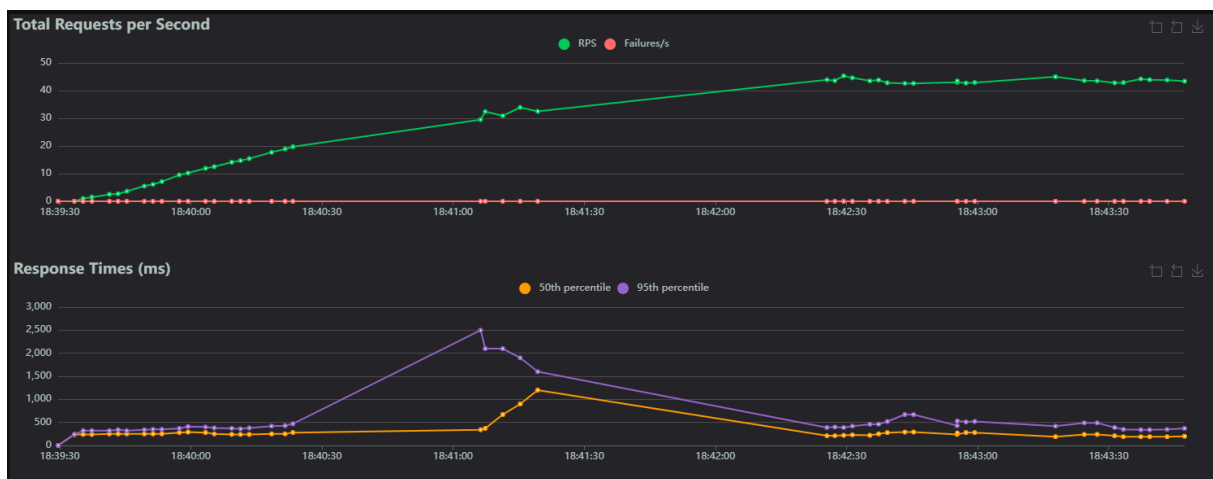


Figura 41: 100 Threads

4.2.2) 500 threads

Com 500 *threads*, começamos a observar sinais mais evidentes de saturação na infraestrutura da aplicação. Embora o sistema tenha conseguido escalar o número de requisições por segundo (RPS), conforme representado pela linha verde no gráfico superior, os tempos de resposta aumentaram significativamente, com o percentil 95 ultrapassando os 9.000ms em alguns momentos.

Este aumento acentuado na latência, visível no gráfico inferior, sugere que os recursos disponíveis — em especial a base de dados — começaram a atingir os seus limites. Mesmo com o *auto-scaling* ativado no *App Engine* e no *Cloud Run*, a instância única e modesta do *Cloud SQL* torna-se um *bottleneck* neste cenário, impedindo uma resposta eficiente à elevada concorrência.

Apesar disso, é importante destacar que a taxa de falhas (linha vermelha) manteve-se praticamente nula, indicando que o sistema continuou funcional, mesmo sob pressão. No entanto, a experiência do utilizador começaria a ser afetada negativamente, principalmente em operações que dependem de interações com a base de dados.

Estes resultados reforçam a necessidade de considerar o redimensionamento da base de dados ou o uso de mecanismos de cache/distribuição de carga para sustentar um maior volume de utilizadores simultâneos em produção.

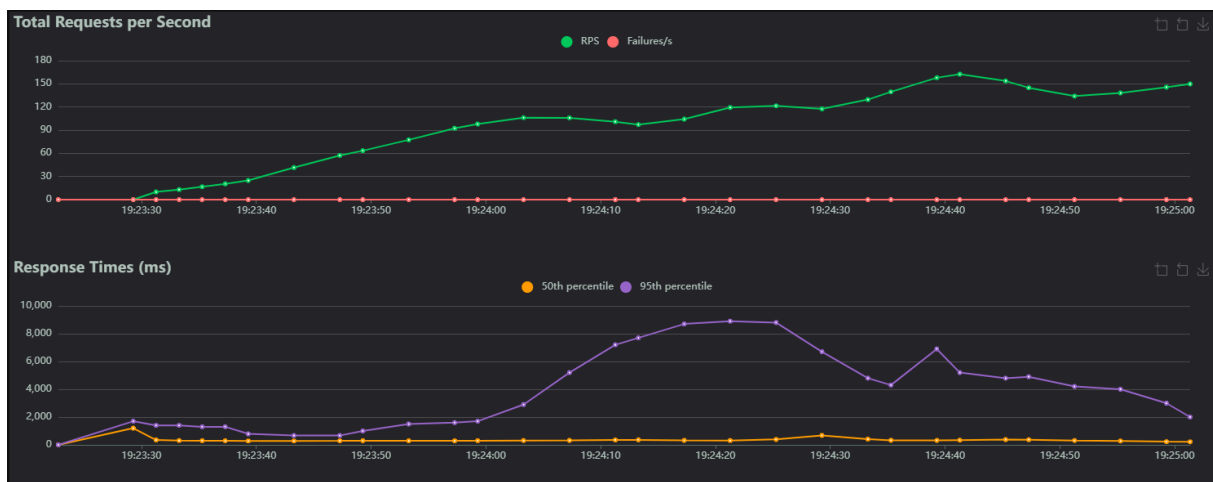


Figura 42: 500 Threads

4.2.3) 1500 threads

Com o aumento para 1500 *threads*, e uma rampa de 50 utilizadores por segundo, a infraestrutura começou a evidenciar sinais mais claros de saturação. Inicialmente, o sistema respondeu de forma positiva, atingindo um pico de cerca de 270 pedidos por segundo (RPS), conforme representado pela linha verde no gráfico superior. No entanto, pouco depois, observa-se uma quebra acentuada no RPS, o que indica que foram atingidos os limites da capacidade de resposta.

Ao nível dos tempos de resposta, a degradação é bastante evidente: o percentil 95 ultrapassa os 45.000 ms, enquanto o percentil 50 se aproxima dos 20.000 ms, sinal de que a maioria dos utilizadores experienciou atrasos significativos. Apesar disso, a taxa de falhas (linha vermelha) manteve-se praticamente nula, o que demonstra que o sistema continuou a aceitar os pedidos, embora com tempos de espera muito elevados.

Estes resultados deixam claro que, nesta fase, a principal limitação do sistema está ao nível da base de dados, cuja configuração mais modesta impede um escalamento eficaz. Mesmo com o *auto-scaling* ativo no *frontend* e *backend*, a aplicação deixa de conseguir acompanhar o crescimento da carga de forma eficiente.

Este teste marca, assim, um ponto de viragem: a partir deste volume de carga, não basta apenas escalar horizontalmente — é necessário otimizar os componentes críticos, nomeadamente a base de dados, para garantir a estabilidade e a performance em cenários de elevada concorrência. Os testes seguintes permitirão avaliar se é possível recuperar capacidade de resposta ou se a degradação se agrava com cargas ainda superiores.

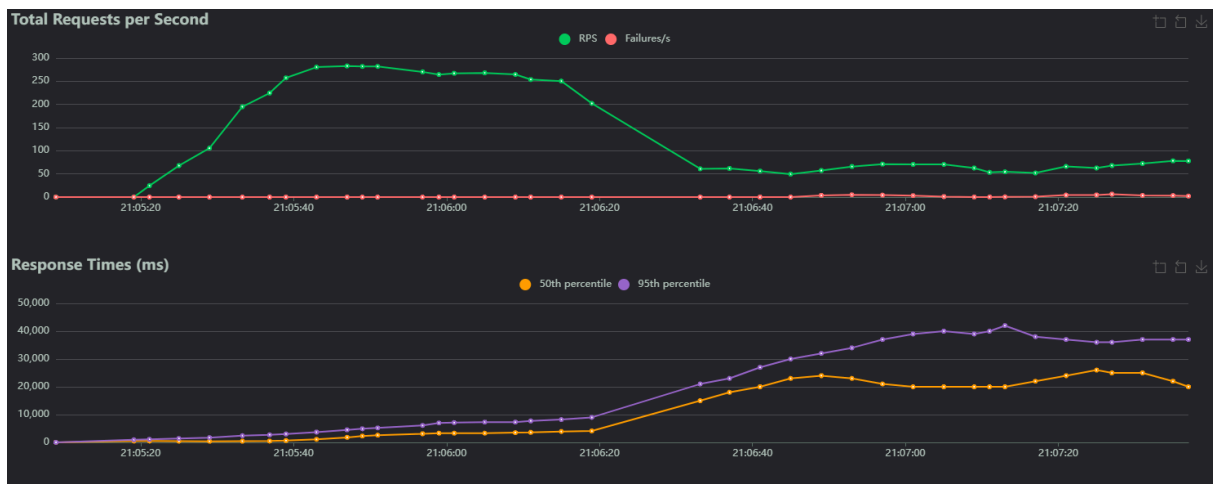


Figura 43: 1500 Threads

4.2.4) 3000 threads

Com 3000 *threads* e uma rampa de 50 utilizadores por segundo, a aplicação enfrentou um cenário de carga bastante elevado, revelando limitações mais evidentes na sua capacidade de resposta. O número de requisições por segundo (RPS) atingiu um pico próximo dos 250, mas rapidamente entrou em declínio, estabilizando depois num patamar mais baixo. Este comportamento indica que o sistema atingiu um ponto de saturação e não conseguiu manter o mesmo ritmo de processamento à medida que a carga aumentava.

A taxa de falhas (linha vermelha no gráfico superior) aumentou de forma gradual, sugerindo que o sistema começou a rejeitar pedidos ou a ultrapassar os limites de tempo de resposta aceitáveis. Esta tendência reflete-se claramente nos tempos de resposta (gráfico inferior), com o percentil 95 a ultrapassar os 60.000 ms e o percentil 50 a manter-se acima dos 30.000 ms durante boa parte do teste.

Estes resultados confirmam que, a partir deste nível de concorrência, a infraestrutura deixa de conseguir acompanhar a carga, mesmo com *auto-scaling* ativo. A base de dados, configurada com recursos limitados, continua a ser o principal fator de estrangulamento, mas é possível que também existam impactos cumulativos ao nível do *backend* e da gestão de filas.

A performance degradada, acompanhada por um aumento nas falhas, reforça a necessidade de rever a arquitetura da solução caso se pretenda escalar para cenários com milhares de utilizadores concorrentes. O próximo teste, com 4000 *threads*, permitirá perceber se esta degradação estabiliza ou se se agrava de forma crítica.



Figura 44: 3000 Threads

4.2.5) 4000 threads

No teste mais exigente, com 4000 *threads* e uma rampa de 50 utilizadores por segundo, a aplicação demonstrou os limites máximos da sua arquitetura atual. O número de requisições por segundo (RPS) atingiu um pico superior a 270, mas sofreu uma quebra abrupta pouco depois, estabilizando em valores significativamente mais baixos. Simultaneamente, observa-se um aumento acentuado na taxa de falhas (linha vermelha), que se manteve elevada ao longo do resto do teste.

Este comportamento reflete um ponto de saturação claro, em que a infraestrutura já não consegue processar nem enfileirar todas as requisições. A componente da base de dados, que ao longo dos testes anteriores já se revelava um *bottleneck*, aqui entra em colapso funcional, impossibilitando o sistema de manter a carga.

Os tempos de resposta também atingem valores extremos: o percentil 95 aproxima-se dos 90.000 ms, e o percentil 50 chega a ultrapassar os 50.000 ms, tornando a experiência do utilizador impraticável. É importante notar que há ainda um comportamento irregular nos tempos de resposta (com quebras e oscilações), o que pode indicar instabilidade provocada pela tentativa de escalamento forçado ou reinício de instâncias sobrecarregadas.

Este teste evidencia de forma definitiva que, para suportar este nível de concorrência, são necessárias mudanças estruturais: reforço da capacidade da base de dados, utilização de mecanismos de cache, partição de carga, e uma estratégia de escalamento mais distribuída e resiliente. Sem estas melhorias, o sistema não conseguirá escalar de forma eficiente para cargas elevadas, limitando o seu uso em cenários reais com muitos utilizadores simultâneos.

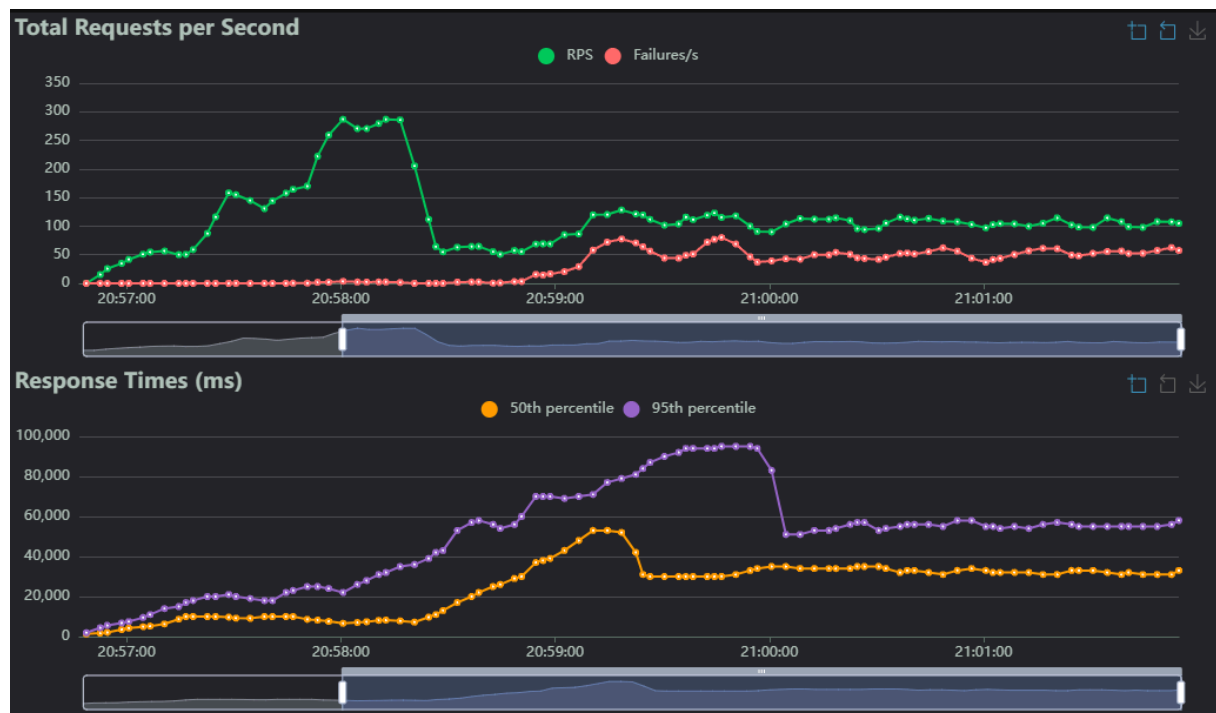


Figura 45: 4000 Threads

4.2.6) Conclusão dos testes de carga

A realização dos testes de carga permitiu avaliar de forma progressiva a capacidade da aplicação para lidar com diferentes níveis de concorrência. Nos primeiros cenários (até 500 *threads*), o sistema demonstrou um desempenho estável, com tempos de resposta aceitáveis e sem registo significativo de falhas. Esta estabilidade confirma que a arquitetura atual, com *auto-scaling* no *frontend* (*App Engine*) e *backend* (*Cloud Run*), é eficaz para cargas moderadas.

Contudo, a partir dos 1500 utilizadores simultâneos, começaram a surgir sintomas claros de saturação: tempos de resposta elevados, quedas no número de requisições por segundo e aumento da latência, ainda que com taxas de falhas reduzidas. Estes sinais agravaram-se nos testes com 3000 e 4000 *threads*, nos quais se verificou não só uma degradação significativa da performance, como também um aumento acentuado nas falhas, comprometendo a fiabilidade da aplicação.

A análise dos resultados aponta para a base de dados (*Google Cloud SQL*), configurada com recursos limitados, como principal *bottleneck* da arquitetura. O escalamento automático do *frontend* e *backend* revela-se eficaz até certo ponto, mas deixa de ter impacto quando a base de dados atinge os seus limites de capacidade.

Em suma, a aplicação apresenta uma base sólida para operar com cargas médias, mas carece de melhorias estruturais para suportar cenários de elevada concorrência. Para evoluir nesse sentido, será essencial reforçar a camada de dados, adotar estratégias de cache, otimizar o acesso a recursos partilhados e, eventualmente, repensar a divisão de responsabilidades entre os serviços. Estas otimizações serão fundamentais para garantir escalabilidade, estabilidade e uma boa experiência do utilizador em contextos de produção com maior exigência.

4.3) Usabilidade

Para analisar a usabilidade do sistema decidimos utilizar os métodos analíticos de **Avaliação Heurística** e **Cognitive Walkthrough**. Com estes métodos tentamos então prever potenciais problemas de usabilidade apenas com a equipa de desenvolvimento.

4.3.1) Avaliação Heurística

Avaliação heurística é uma técnica de avaliação de interfaces para identificar problemas de usabilidade. Para a avaliação, são utilizadas um conjunto de heurísticas para guiar a análise, tendo o nosso grupo decidido utilizar as heurísticas de Nielsen para nos auxiliar.

Visibilidade do estado do sistema

Perante a realização de tarefas no sistema, o utilizador é informado do resultado das operações através de mensagens que oferecem *feedback* ao utilizador, que aparecem no ecrã brevemente para o notificar do resultado da operação realizada, como podemos visualizar no canto superior direito da fig.46. Também é utilizada uma componente “*Loading*” para informar o utilizador quando algum pedido é recebido e está à espera de resposta.

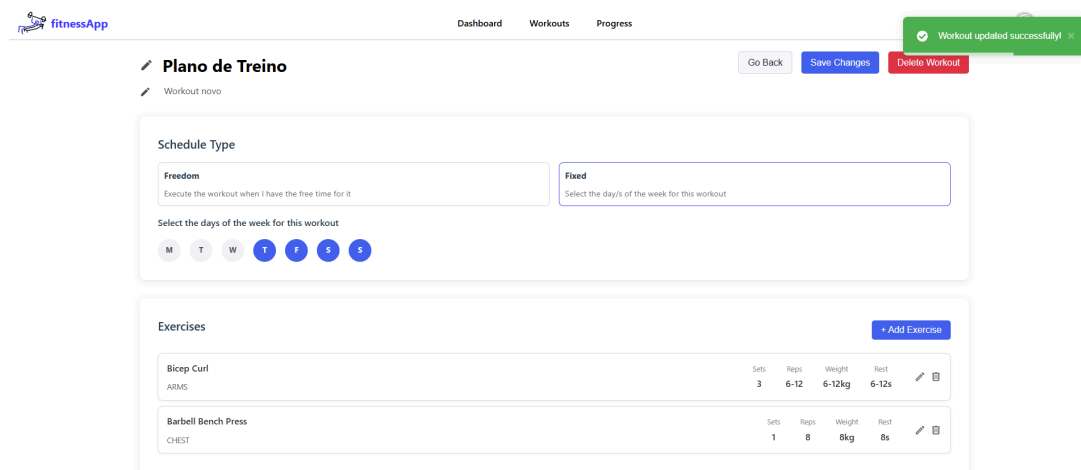


Figura 46: Demonstração de mensagens de *feedback*

Correspondência entre o sistema e o mundo real

Na interface é utilizada linguagem familiar ao utilizador e que é comum no ambiente em questão (ginásios). Alguns dos termos utilizados mais reconhecíveis são: *Sets*, *Workouts*, *Reps* (repetições), etc. É permitido também ao utilizador alternar entre a utilização de unidades métricas ou unidades imperiais, adaptando assim o sistema às convenções culturais ou regionais do utilizador.

Controlo e liberdade do utilizador

A interface desenvolvida permite ao utilizador navegar de forma livre, sem exigir nenhuma ordem ao utilizador. Para o utilizador saber onde se encontra na *Navbar*, a página é identificada com uma cor diferente das restantes e uma linha por baixo do nome, como podemos visualizar na fig.46.

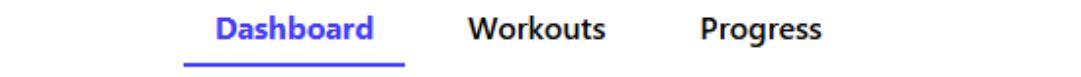


Figura 47: Identificação da localização na *Navbar*

Também são disponibilizadas ao utilizador, ao longo da interface, saídas através de um botão de retorno. Por exemplo, na página de edição de treinos representada na fig.48, onde mostra que existe um botão para voltar à página anterior, ou também nos *pop ups* de confirmação, onde é oferecida a opção de cancelar a ação.

Figura 48: Exemplo botão retorno 1

Create New Workout

What do you want to name this workout?

Create

Cancel

Figura 49: Exemplo botão retorno 2

Consistência e normas

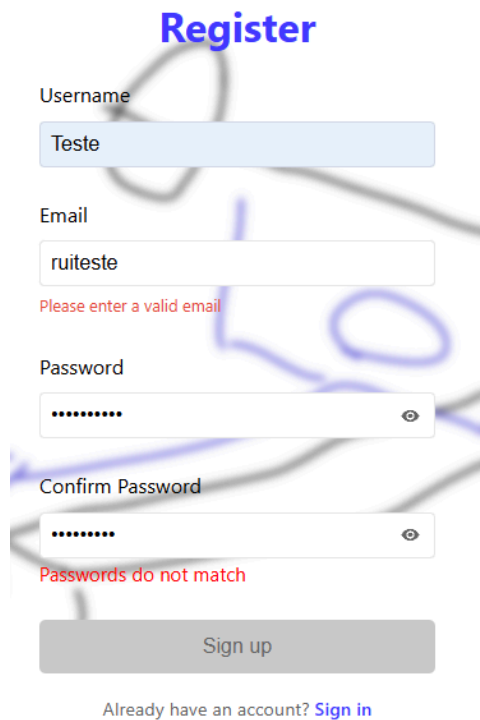
Tivemos em conta a utilização de ícones comuns e a que os utilizadores já estão habituados noutras aplicações, como é o caso do sino para as notificações, do “i” para as ajudas na página dos planos de treino e também das setas para a ordenação ao longo da interface.

Client ID ↓	Client Name ↓	Email ↓	Height ↓	Weight ↓	Latest Activity ↓	Actions
-------------	---------------	---------	----------	----------	-------------------	---------

Figura 50: Ícones padronizados

Prevenção de erros

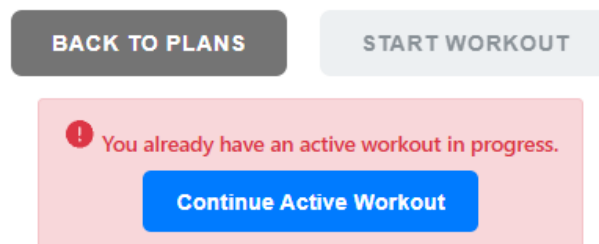
Na interface desenvolvida são implementados mecanismos de prevenção de erros para tentar evitar que os utilizadores cometam erros durante a interação, fornecendo avisos claros e validação de dados. Podemos visualizar como exemplo quando o utilizador insere dados inválidos no registo (na fig.51), ou onde mostra que não permite iniciar um plano de treino, caso esteja um a decorrer, desabilitando o botão (na fig.52).



The image shows a 'Register' form with the following fields and feedback:

- Username:** A text input containing 'Teste'.
- Email:** A text input containing 'ruiteste'. Below it, a red error message reads: 'Please enter a valid email'.
- Password:** A password input with masked characters. To its right is an eye icon for toggling visibility.
- Confirm Password:** A password input with masked characters. To its right is an eye icon for toggling visibility. Below it, a red error message reads: 'Passwords do not match'.
- Sign up:** A grey button at the bottom of the form.
- Footer:** A link that says 'Already have an account? [Sign in](#)'.

Figura 51: Prevenção de erros no registo



The image shows a UI element for starting a workout with the following components:

- Buttons:** Two buttons at the top: 'BACK TO PLANS' (dark grey) and 'START WORKOUT' (light grey).
- Error Message:** A pink rectangular box containing a red exclamation mark icon and the text: 'You already have an active workout in progress.'
- Action Button:** A blue button with the text 'Continue Active Workout' located inside the pink error box.

Figura 52: Prevenção de erros a iniciar plano de treino

Reconhecer em vez de recordar

No nosso sistema não existe qualquer contexto em que o utilizador necessita de informação que apenas existe em páginas anteriores, sendo que qualquer página apresenta tudo aquilo que o utilizador precisa de saber para a navegar. Um exemplo de uma medida que tomamos para o assegurar foi durante a execução de um *workout*, em que é fornecida ao utilizador a informação sobre os *sets* e pesos planeados para aquele treino, permitindo que o utilizador não tenha que recordar esses detalhes.

Flexibilidade e eficiência de utilização

De forma a permitir a utilizadores experientes acelerar a sua interação com o sistema, a interface desenvolvida oferece vários atalhos que tornam a experiência de quem a usa regularmente mais eficiente e agradável.

Existem, por exemplo, diversas formas de chegar à página de execução de um *workout*, quer seja através do calendário (clicando no *workout* que deseja iniciar), ou da secção que demonstra o próximo plano de treino planeado, como podemos ver na fig.53. Esta secção têm também um atalho no caso de existir um treino a decorrer, como demonstrado na fig.54 e na fig.52.

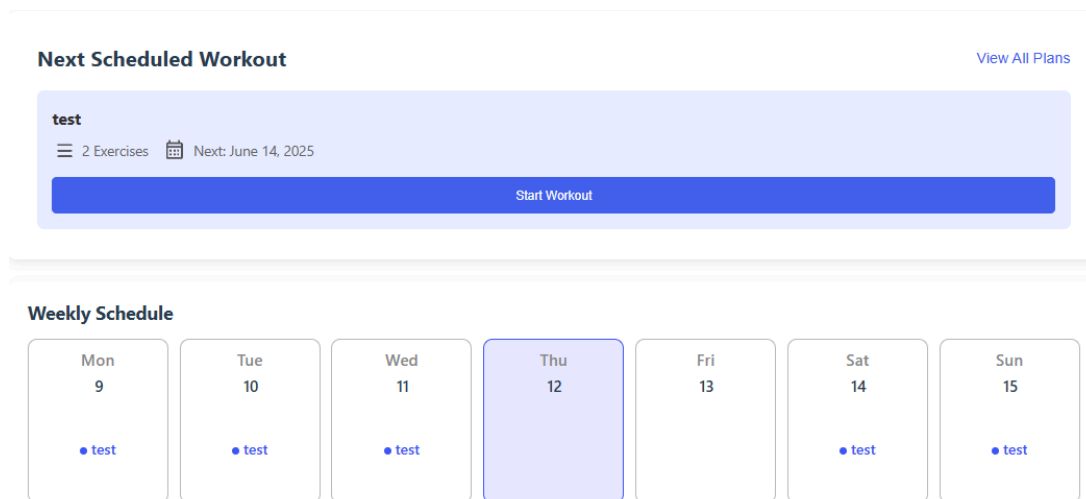


Figura 53: Prevenção de erros a iniciar plano de treino

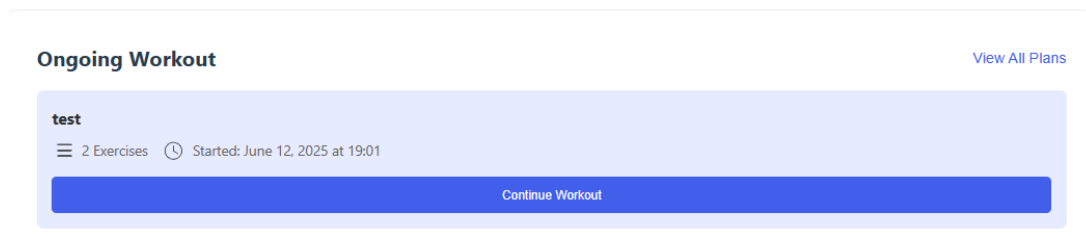


Figura 54: Prevenção de erros a iniciar plano de treino

Para além disso, existem também várias opções de filtragem em componentes cujo tamanho pode chegar a ser bastante extensivo, como é o caso da biblioteca de exercícios e do histórico dos *workouts* executados.

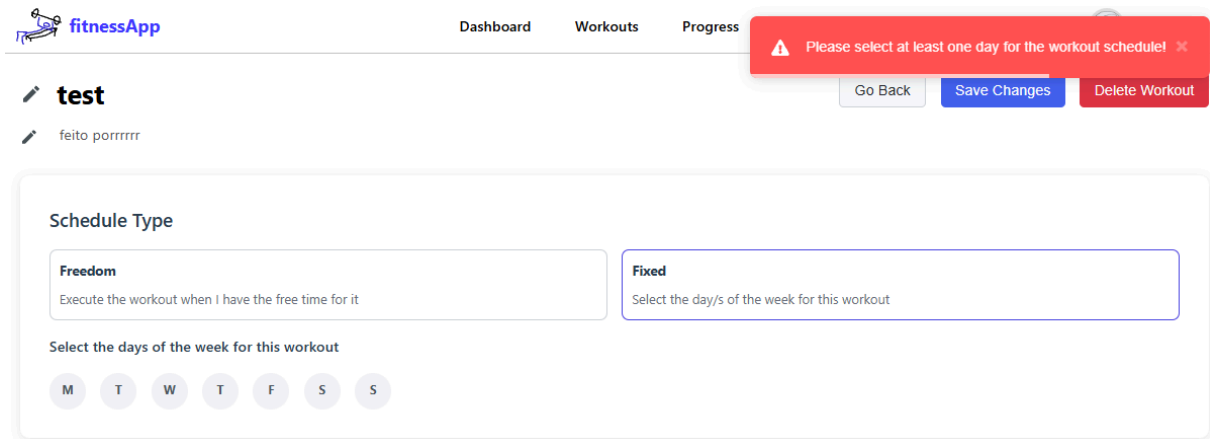
Por fim, é também de interesse referir que o logótipo da aplicação, ao ser clicado, leva o utilizador à *dashboard*.

Desenho estético e minimalista

O desenho do sistema é bastante simples e minimalista, onde apresentamos apenas a informação necessária para que o utilizador consiga interagir de forma eficiente, sem distrações ou elementos que o impeçam de alcançar os seus objetivos na aplicação.

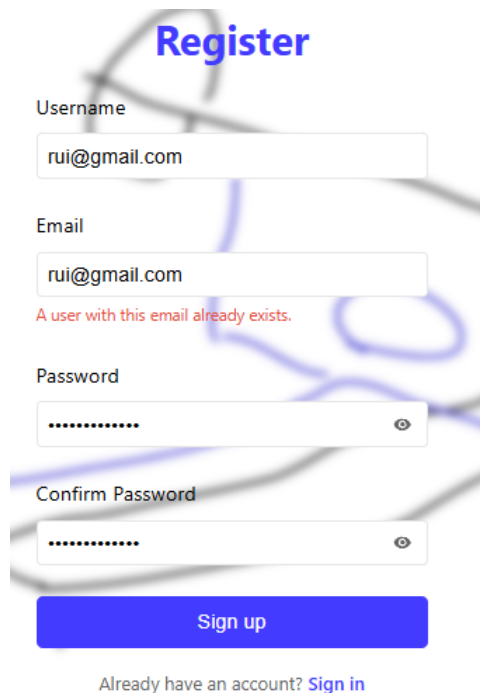
Ajudar os utilizadores a reconhecer, diagnosticar e recuperar de erros

Na interface são apresentadas mensagens de erro informativas para indicar ao utilizador o erro ocorrido. Podemos ver exemplos disso nas três figuras em baixo: no caso da fig.55, em que o utilizador tenta guardar com o tipo de horário fixo, mas sem seleccionar qualquer dia; no caso da fig.56, em que o utilizador tenta realizar o registo com um email já existente; no caso da fig.57, em que o utilizador tenta registar um peso negativo.



The screenshot shows the 'fitnessApp' interface. At the top, there's a navigation bar with 'Dashboard', 'Workouts', and 'Progress'. A red error banner at the top right says: 'Please select at least one day for the workout schedule!'. Below the banner, there are buttons: 'Go Back', 'Save Changes', and 'Delete Workout'. The main content area is titled 'test' and 'feito porrrrrr'. Under 'Schedule Type', there are two options: 'Freedom' (selected) and 'Fixed'. The 'Fixed' option is highlighted with a blue border. Below the options, there's a section 'Select the days of the week for this workout' with seven circular buttons labeled M, T, W, T, F, S, S. The 'Fixed' option's description says 'Select the day/s of the week for this workout'.

Figura 55: Erro em caso de não serem seleccionados dias para os planos



The screenshot shows a 'Register' form. It has fields for 'Username', 'Email', 'Password', and 'Confirm Password'. The 'Email' field contains 'rui@gmail.com'. Below the 'Email' field, there's a red error message: 'A user with this email already exists.' At the bottom, there's a blue 'Sign up' button and a link 'Already have an account? Sign in'.

Figura 56: Erro em caso de registo com email já existente

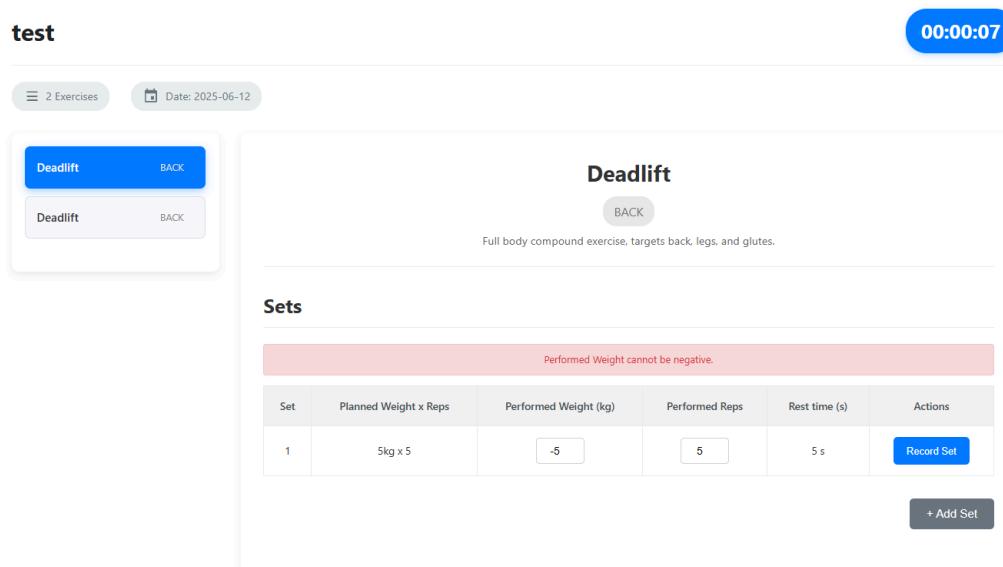


Figura 57: Erro em tentar inserir um peso negativo

Ajuda e documentação

A disponibilização de ajudas na interface é um aspeto bastante importante para permitir que novos utilizadores consigam utilizar a interface de forma eficaz. Alguns exemplos na interface desenvolvida seriam os ícones de ajuda na distinção entre planos ativos e inativos, demonstrados na fig.25 e fig26, assim como a distinção feita entre horário fixo e horário livre, demonstrado na fig.55.

4.3.2) Cognitive Walkthrough

Cognitive Walkthrough é um método analítico para análise pensado para avaliar sistemas utilizados sem treino prévio, ou seja, é a análise à experiência de um utilizador que interage com o sistema pela primeira vez, o que permite descobrir e prever problemas de usabilidade.

Para utilizar este método devem ser cumpridos alguns requisitos, como descrever quem são os utilizadores, para identificar conhecimentos prévios e habilidades técnicas, assim como definir as principais tarefas a ser executadas no sistema.

Para cada tarefa, o utilizador vai respondendo a um conjunto pré definido de questões:

1. A ação correta é suficientemente evidente para o utilizador?
2. O controlo para executar a ação está visível?
3. Irá o utilizador associar a ação correta ao controlo?
4. Irá o utilizador interpretar de forma correta a resposta do sistema à ação escolhida? (saberá se fez a escolha certa?)

O utilizador deve responder às perguntas enunciadas em cima com “Sim” ou “Não”, e cada resposta negativa deve ser classificada por severidade:

Tipo 1: o problema pode causar confusão ou demora na execução da tarefa.

Tipo 2: o problema pode impedir que o utilizador consiga executar a tarefa sem ajuda.

Tipo 3: o problema impede a execução da tarefa.

Para conseguirmos então realizar estes testes decidimos utilizar três tarefas principais: criar um plano

de treino, executar um treino e o professor verificar o progresso de aluno. Os questionários foram realizados por 2 membros da equipa de desenvolvimento, tendo ambos conhecimentos experiência na área de informática. Os resultados obtidos estão demonstrados nas tabelas seguintes:

Sistema				
Tarefa	Criar um plano de treino			
Utilizador				
Passos	Questões	OK (S/N)	Risco (1-3)	Problema/Sugestão
Clicar em Create new workout	1	S		
	2	S		
	3	S		
	4	S		
Escolher Tipo de Horário	1	S		
	2	S		
	3	S		
	4	S		
Caso seja horário fixo, escolher os dias	1	S		
	2	S		
	3	S		
	4	S		
Adicionar Exercícios e os sets	1	S		
	2	S		
	3	S		
	4	S		
Salvar	1	S		
	2	S		
	3	S		
	4	S		

Figura 58: Utilizador 1 - Criar Plano de Treino

Sistema				
Tarefa	Criar um plano de treino			
Utilizador				
Passos	Questões	OK (S/N)	Risco (1-3)	Problema/Sugestão
Clicar em Create new workout	1	S		
	2	S		
	3	S		
	4	S		
Escolher Tipo de Horário	1	S		
	2	S		
	3	S		
	4	S		
Caso seja horário fixo, escolher os dias	1	S		
	2	S		
	3	S		
	4	S		
Adicionar Exercícios e os sets	1	S		
	2	S		
	3	S		
	4	S		
Salvar	1	S		
	2	S		
	3	S		
	4	S		

Figura 59: Utilizador 2 - Criar Plano de Treino

Sistema				
Tarefa	Executar Plano de Treino			
Utilizador				
Passos	Questões	OK (S/N)	Risco (1-3)	Problema/Sugestão
Escolher um plano de treino clicando em Go To Workout	1	S	S	
	2	S	S	
	3	S	S	
	4	S	S	
Clicar em Start Workout	1	S	S	
	2	S	S	
	3	S	S	
	4	S	S	
Registar sets do exercício	1	S	S	
	2	S	S	
	3	S	S	
	4	S	S	
Completar plano ao clicar em Complete workout	1	S	S	
	2	S	S	
	3	S	S	
	4	S	S	

Figura 60: Utilizador 1 - Executar Plano de Treino

Sistema				
Tarefa	Executar Plano de Treino			
Utilizador				
Passos	Questões	OK (S/N)	Risco (1-3)	Problema/Sugestão
Escolher um plano de treino clicando em Go To Workout	1	S	S	
	2	S	S	
	3	S	S	
	4	S	S	
Clicar em Start Workout	1	S	S	
	2	S	S	
	3	S	S	
	4	S	S	
Registar sets do exercício	1	N		Utilizador pode demorar um pouco a entender onde deve registar os sets
	2	S		
	3	S		
	4	S		
Completar plano ao clicar em Complete workout	1	S		
	2	S		
	3	S		
	4	S		

Figura 61: Utilizador 2 - Executar Plano de Treino

Sistema				
Tarefa	Professor consultar progresso de cliente			
Utilizador				
Passos	Questões	OK (S/N)	Risco (1-3)	Problema/Sugestão
Navegar para a secção My Clients	1	S		
	2	S		
	3	S		
	4	S		
Selecionar a opção View Info de um utilizador	1	N	1	Utilizador pode não achar o texto do botão esclarecedor suficiente
	2	S		
	3	S		
	4	S		
Clicar em Check Client Progress	1	S		
	2	S		
	3	S		
	4	S		
Verificar o progresso de um plano	1	S		
	2	S		
	3	S		
	4	S		

Figura 62: Utilizador 1 - Consultar Progresso de Cliente

Sistema				
Tarefa	Professor consultar progresso de cliente			
Utilizador				
Passos	Questões	OK (S/N)	Risco (1-3)	Problema/Sugestão
Navegar para a secção My Clients	1	S		
	2	S		
	3	S		
	4	S		
Selecionar a opção View Info de um utilizador	1	S		
	2	S		
	3	S		
	4	S		
Clicar em Check Client Progress	1	S		
	2	S		
	3	S		
	4	S		
Verificar o progresso de um plano	1	S		
	2	S		
	3	S		
	4	S		

Figura 63: Utilizador 2 - Consultar Progresso de Cliente

Com base nos resultados obtidos, é possível visualizar que os utilizadores não encontraram dificuldades a utilizar a aplicação, existindo apenas dois pontos onde o utilizador pode ter mais dificuldades. No entanto não é possível tirar uma conclusão objetiva deste teste devido ao número reduzido de utilizadores que o realizou.

5) Conclusão

Ao realizar este trabalho e documentar o seu desenvolvimento, conseguimos consolidar os conceitos lecionados ao explorar um caso prático e utilizar os conceitos lecionados.

A aplicação foi desenvolvida tendo em conta as necessidades dos utilizadores, oferecendo uma solução funcional e prática. As interfaces foram desenvolvidas para garantir uma experiência intuitiva e de utilização simples.

Para trabalho futuro gostaríamos de realizar algumas melhorias à nossa aplicação tais como: melhorar a segurança em termos de acessos aos dados da aplicação, implementar *templates* para os professores poderem utilizar na criação de planos para os seus alunos e por fim desenvolver mais a biblioteca de exercícios, permitindo aos utilizadores criar exercícios personalizados para a sua biblioteca.