🏃 Gestão de Atletas - Documentação Técnica

📋 Visão Geral

O módulo de Gestão de Atletas é o núcleo do sistema Personal Trainer Pro, responsável por todo o CRUD (Create, Read, Update, Delete) dos atletas, acompanhamento de progresso, métricas de desempenho e histórico de atividades.

🏗️ Arquitetura do Módulo

Componentes Principais

```
athletes/
├── AthleteList.js          # Lista principal de atletas
├── AthleteCard.js          # Card individual do atleta
├── AthleteDetail.js        # Página de detalhes completos
├── AthleteForm.js          # Formulário de criação/edição
├── AthleteMetrics.js       # Métricas e estatísticas
├── AthleteProgress.js      # Gráficos de progresso
├── AthleteSearch.js        # Busca e filtros
└── AthleteCheckin.js       # Check-ins diários
```

📊 Modelo de Dados

Estrutura do Atleta

javascript

```javascript
const athleteSchema = {
  id: String,                    // UUID único
  personalInfo: {
    name: String,                // Nome completo
    email: String,               // Email único
    phone: String,               // Telefone
    dateOfBirth: Date,           // Data de nascimento
    gender: String,              // 'male' | 'female' | 'other'
    profilePhoto: String,        // URL da foto
    emergencyContact: {
      name: String,
      phone: String,
      relationship: String
    }
  },
  healthInfo: {
    height: Number,              // Altura em cm
    currentWeight: Number,       // Peso atual em kg
    targetWeight: Number,        // Peso objetivo em kg
    medicalConditions: Array,    // Condições médicas
    medications: Array,          // Medicamentos
    allergies: Array,            // Alergias
    injuries: Array,             // Histórico de lesões
    fitnessLevel: String,        // 'beginner' | 'intermediate' | 'advanced'
  },
  goals: {
    primary: String,             // Objetivo principal
    secondary: Array,            // Objetivos secundários
    targetDate: Date,            // Data objetivo
    description: String          // Descrição detalhada
  },
  membership: {
    startDate: Date,             // Data de início
    plan: String,                // Tipo de plano
    status: String,              // 'active' | 'inactive' | 'suspended'
    paymentStatus: String,       // 'current' | 'overdue' | 'pending'
    monthlyFee: Number           // Mensalidade
  },
  metrics: {
    attendance: {
      total: Number,             // Total de presenças
      percentage: Number,        // Percentual de frequência
      streak: Number,            // Sequência atual
      lastWorkout: Date          // Último treino
    }
```

```
  },
  progress: {
    weightLoss: Number,       // Perda de peso em kg
    muscleGain: Number,       // Ganho muscular em kg
    bodyFatReduction: Number, // Redução de gordura em %
    strengthGains: Object     // Ganhos de força por exercício
  },
  measurements: {
    chest: Number,            // Peito em cm
    waist: Number,            // Cintura em cm
    hips: Number,             // Quadril em cm
    arms: Number,             // Braços em cm
    thighs: Number,           // Coxas em cm
    recordedAt: Date          // Data da medição
  }
 },
 checkins: Array,             // Check-ins diários
 workoutHistory: Array,       // Histórico de treinos
 notes: Array,                // Observações do PT
 createdAt: Date,
 updatedAt: Date
}
```

Estrutura de Check-in Diário

javascript

```
const checkinSchema = {
 id: String,
 date: Date,
 mood: String,               // 'great' | 'good' | 'neutral' | 'bad' | 'terrible'
 energy: Number,             // 1-10
 sleep: Number,              // 1-10 (horas de sono)
 motivation: Number,         // 1-10
 stress: Number,             // 1-10
 nutrition: Number,          // 1-10
 hydration: Number,          // 1-10
 soreness: Number,           // 1-10
 notes: String,              // Observações do atleta
 weight: Number,             // Peso do dia (opcional)
 completedWorkout: Boolean,  // Se treinou no dia
 workoutRating: Number       // Avaliação do treino (1-10)
}
```

🔧 Hooks Personalizados

useAthletes

javascript

```javascript
// hooks/useAthletes.js
import { useState, useEffect, useCallback } from 'react';
import { athleteService } from '../services/api/athletes';

export const useAthletes = (filters = {}) => {
  const [athletes, setAthletes] = useState([]);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState(null);
  const [pagination, setPagination] = useState({
    page: 1,
    limit: 10,
    total: 0
  });

  // Buscar atletas
  const fetchAthletes = useCallback(async () => {
    try {
      setLoading(true);
      const response = await athleteService.getAll(filters, pagination);
      setAthletes(response.data);
      setPagination(response.pagination);
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  }, [filters, pagination.page, pagination.limit]);

  // Adicionar atleta
  const addAthlete = async (athleteData) => {
    try {
      const newAthlete = await athleteService.create(athleteData);
      setAthletes(prev => [newAthlete, ...prev]);
      return { success: true, data: newAthlete };
    } catch (err) {
      return { success: false, error: err.message };
    }
  };

  // Atualizar atleta
  const updateAthlete = async (id, athleteData) => {
    try {
      const updatedAthlete = await athleteService.update(id, athleteData);
      setAthletes(prev =>
```

```
      prev.map(athlete =>
        athlete.id === id ? updatedAthlete : athlete
      )
    );
    return { success: true, data: updatedAthlete };
  } catch (err) {
    return { success: false, error: err.message };
  }
};

// Remover atleta
const removeAthlete = async (id) => {
  try {
    await athleteService.delete(id);
    setAthletes(prev => prev.filter(athlete => athlete.id !== id));
    return { success: true };
  } catch (err) {
    return { success: false, error: err.message };
  }
};

// Buscar atleta por ID
const getAthleteById = useCallback(async (id) => {
  try {
    return await athleteService.getById(id);
  } catch (err) {
    throw new Error(err.message);
  }
}, []);

useEffect(() => {
  fetchAthletes();
}, [fetchAthletes]);

return {
  athletes,
  loading,
  error,
  pagination,
  actions: {
    addAthlete,
    updateAthlete,
    removeAthlete,
    getAthleteById,
    refetch: fetchAthletes
  }
};
```

```
};
```

useAthleteProgress

javascript

```javascript
// hooks/useAthleteProgress.js
export const useAthleteProgress = (athleteId) => {
  const [progressData, setProgressData] = useState(null);
  const [loading, setLoading] = useState(true);

  const calculateProgress = useCallback((athlete) => {
    const {
      startWeight = athlete.healthInfo.currentWeight,
      targetWeight = athlete.healthInfo.targetWeight,
      currentWeight = athlete.healthInfo.currentWeight
    } = athlete.metrics?.progress || {};

    const weightProgress = Math.abs(currentWeight - startWeight) /
                           Math.abs(targetWeight - startWeight) * 100;

    const attendanceRate = athlete.metrics?.attendance?.percentage || 0;

    const overallProgress = (weightProgress + attendanceRate) / 2;

    return {
      weight: {
        start: startWeight,
        current: currentWeight,
        target: targetWeight,
        progress: Math.min(weightProgress, 100)
      },
      attendance: attendanceRate,
      overall: Math.min(overallProgress, 100)
    };
  }, []);

  return { progressData, loading, calculateProgress };
};
```

🔄 Serviços de API

Athlete Service

javascript

```javascript
// services/api/athletes.js
import { apiClient } from './base';

export const athleteService = {
  // Buscar todos os atletas
  async getAll(filters = {}, pagination = {}) {
    const params = new URLSearchParams({
      ...filters,
      page: pagination.page || 1,
      limit: pagination.limit || 10
    });

    const response = await apiClient.get(`/athletes?${params}`);
    return response.data;
  },

  // Buscar atleta por ID
  async getById(id) {
    const response = await apiClient.get(`/athletes/${id}`);
    return response.data;
  },

  // Criar novo atleta
  async create(athleteData) {
    const response = await apiClient.post('/athletes', athleteData);
    return response.data;
  },

  // Atualizar atleta
  async update(id, athleteData) {
    const response = await apiClient.put(`/athletes/${id}`, athleteData);
    return response.data;
  },

  // Deletar atleta
  async delete(id) {
    await apiClient.delete(`/athletes/${id}`);
  },

  // Adicionar check-in
  async addCheckin(athleteId, checkinData) {
    const response = await apiClient.post(
      `/athletes/${athleteId}/checkins`,
      checkinData
    );
```

```
    );
    return response.data;
  },

  // Buscar métricas do atleta
  async getMetrics(athleteId, period = '30d') {
    const response = await apiClient.get(
      `/athletes/${athleteId}/metrics?period=${period}`
    );
    return response.data;
  },

  // Upload de foto
  async uploadPhoto(athleteId, photoFile) {
    const formData = new FormData();
    formData.append('photo', photoFile);

    const response = await apiClient.post(
      `/athletes/${athleteId}/photo`,
      formData,
      {
        headers: {
          'Content-Type': 'multipart/form-data'
        }
      }
    );
    return response.data;
  }
};
```

✅ Validações

Validação do Formulário de Atleta

javascript

```javascript
// services/validation/athleteValidation.js
import * as yup from 'yup';

export const athleteValidationSchema = yup.object({
  personalInfo: yup.object({
    name: yup
      .string()
      .required('Nome é obrigatório')
      .min(2, 'Nome deve ter pelo menos 2 caracteres')
      .max(100, 'Nome deve ter no máximo 100 caracteres'),

    email: yup
      .string()
      .required('Email é obrigatório')
      .email('Email deve ser válido'),

    phone: yup
      .string()
      .required('Telefone é obrigatório')
      .matches(/^[0-9+\-\s()]+$/, 'Telefone deve ser válido'),

    dateOfBirth: yup
      .date()
      .required('Data de nascimento é obrigatória')
      .max(new Date(), 'Data de nascimento não pode ser futura')
      .test('age', 'Atleta deve ter pelo menos 16 anos', function(value) {
        const today = new Date();
        const birthDate = new Date(value);
        const age = today.getFullYear() - birthDate.getFullYear();
        return age >= 16;
      }),

    gender: yup
      .string()
      .required('Género é obrigatório')
      .oneOf(['male', 'female', 'other'], 'Género deve ser válido')
  }),

  healthInfo: yup.object({
    height: yup
      .number()
      .required('Altura é obrigatória')
      .min(100, 'Altura deve ser pelo menos 100cm')
      .max(250, 'Altura deve ser no máximo 250cm'),
```

```
    currentWeight: yup
      .number()
      .required('Peso atual é obrigatório')
      .min(30, 'Peso deve ser pelo menos 30kg')
      .max(300, 'Peso deve ser no máximo 300kg'),

    targetWeight: yup
      .number()
      .required('Peso objetivo é obrigatório')
      .min(30, 'Peso objetivo deve ser pelo menos 30kg')
      .max(300, 'Peso objetivo deve ser no máximo 300kg'),

    fitnessLevel: yup
      .string()
      .required('Nível de fitness é obrigatório')
      .oneOf(['beginner', 'intermediate', 'advanced'], 'Nível deve ser válido')
  }),

  goals: yup.object({
    primary: yup
      .string()
      .required('Objetivo principal é obrigatório'),

    targetDate: yup
      .date()
      .min(new Date(), 'Data objetivo deve ser futura')
  })
});

export const validateAthlete = async (athleteData) => {
  try {
    await athleteValidationSchema.validate(athleteData, { abortEarly: false });
    return { isValid: true, errors: {} };
  } catch (error) {
    const errors = {};
    error.inner.forEach(err => {
      errors[err.path] = err.message;
    });
    return { isValid: false, errors };
  }
};
```

🎯 Funcionalidades Principais

1. Listagem de Atletas

- [x] Grid responsivo de cards
- [x] Paginação
- [x] Busca por nome/email
- [x] Filtros por status, plano, nível
- [x] Ordenação por diferentes critérios
- [x] Ações rápidas (editar, ver detalhes, remover)

2. Perfil Detalhado do Atleta

- [x] Informações pessoais completas
- [x] Métricas de progresso
- [x] Histórico de check-ins
- [x] Gráficos de evolução
- [x] Timeline de atividades
- [x] Observações do personal trainer

3. Formulário de Criação/Edição

- [x] Wizard multi-step
- [x] Validações em tempo real
- [x] Upload de foto de perfil
- [x] Auto-save (rascunho)
- [x] Integração com API

4. Check-ins Diários

- [x] Formulário rápido
- [x] Acompanhamento de humor e energia
- [x] Métricas de sono e nutrição
- [x] Histórico visual

5. Análise de Progresso

- [x] Gráficos interativos
- [x] Comparações temporais
- [x] Indicadores de desempenho
- [x] Relatórios exportáveis

🚀 Próximos Passos

1. **Implementar AthleteList** - Componente principal
2. **Criar AthleteForm** - Formulário CRUD
3. **Desenvolver AthleteDetail** - Página de detalhes
4. **Adicionar AthleteMetrics** - Dashboard de métricas
5. **Implementar busca e filtros**
6. **Integrar com API backend**
7. **Adicionar testes automatizados**
8. **Melhorar acessibilidade**

📱 Considerações Mobile

- Layout responsivo em todos os componentes
- Touch-friendly interactions
- Formulários otimizados para mobile
- Performance otimizada para dispositivos lentos
- Funcionalidades offline básicas

♿ Acessibilidade

- ARIA labels em todos os elementos interativos
- Navegação por teclado completa
- Contraste adequado de cores
- Screen reader friendly
- Textos alternativos para imagens

🔒 Segurança

- Validação client-side e server-side
- Sanitização de dados de entrada
- Controle de permissões por papel
- Auditoria de ações realizadas
- Proteção contra CSRF/XSS