🔧 Guia Técnico Detalhado - Maratona Training

📋 Índice

🛠️ Setup do Ambiente

Requisitos

- Node.js 18+
- PostgreSQL 14+
- Yarn (gerenciador de pacotes)
- Git

Instalação Completa

```
# 1. Navegue até o diretório
cd /home/ubuntu/app_maratona/nextjs_space

# 2. Instale todas as dependências
yarn install

# 3. Configure as variáveis de ambiente
cp .env.example .env
# Edite .env com suas credenciais

# 4. Gere o Prisma Client
yarn prisma generate

# 5. Sincronize o schema com o banco
yarn prisma db push

# 6. (Opcional) Popule com dados de teste
yarn prisma db seed

# 7. Inicie o servidor de desenvolvimento
yarn dev
```

Estrutura de .env

```
# Database (obrigatório)
DATABASE_URL="postgresql://user:password@host:port/database?schema=public"

# NextAuth (obrigatório)
NEXTAUTH_SECRET="gere-com-openssl-rand-base64-32"
NEXTAUTH_URL="http://localhost:3000"

# Strava API (obrigatório para integração)
STRAVA_CLIENT_ID="seu-client-id"
STRAVA_CLIENT_SECRET="seu-client-secret"
STRAVA_REDIRECT_URI="http://localhost:3000/api/strava/callback"

# Abacus.AI / OpenAI (obrigatório para IA)
ABACUSAI_API_KEY="seu-api-key"

# Admin (opcional)
ADMIN_EMAIL="admin@example.com"
ADMIN_PASSWORD="admin123"

# Node Environment
NODE_ENV="development"
```

🏗️ Arquitetura de Código

Padrão de Organização

```
Princípios:
1. Componentes reutilizáveis em /components
2. Lógica de negócio em /lib
3. API Routes para backend
4. Prisma para ORM
5. TypeScript para type safety
```

Convenções de Nomenclatura

```
// Arquivos
kebab-case: workout-log-form.tsx
camelCase: aiPlanGenerator.ts

// Componentes React
PascalCase: WorkoutLogForm, StravaConnect

// Funções
camelCase: generatePlan(), calculateVDOT()

// Constantes
UPPER_SNAKE_CASE: MAX_WEEKS, DEFAULT_VDOT

// Tipos/Interfaces
PascalCase: AthleteProfile, TrainingPlan
```

Fluxo de Dados

```
User Action (UI)
    ↓
Component Event Handler
    ↓
API Route (/api/*)
    ↓
Service Function (lib/*)
    ↓
Prisma Client (lib/db.ts)
    ↓
PostgreSQL Database
    ↓
Response → Component → UI Update
```

🧠 Geradores de Plano

ai-plan-generator.ts - Gerador Principal

Localização: `lib/ai-plan-generator.ts`

Função Principal: `generateTrainingPlan()`

```typescript
export async function generateTrainingPlan(
  profile: AthleteProfile,
  raceGoal: RaceGoal
): Promise<GeneratedPlan>
```

Fluxo Interno:

```
1. Validação de Dados
   ↓
   - Verifica se profile existe
   - Verifica se raceGoal existe
   - Valida datas

2. Cálculo de Duração
   ↓
   const today = new Date()
   const raceDate = new Date(raceGoal.raceDate)
   const daysUntilRace = (raceDate - today) / (1000 * 60 * 60 * 24)
   const weeksUntilRace = Math.floor(daysUntilRace / 7)

   // Validações
   if (weeksUntilRace < 8) throw new Error("Mínimo 8 semanas")
   if (weeksUntilRace > 52) throw new Error("Máximo 52 semanas")

3. Definição de Periodização
   ↓
   const periodization = calculatePeriodization(weeksUntilRace, pro-
file.experienceLevel)
   // Retorna: { base, build, peak, taper }

4. Construção do Prompt para IA
   ↓
   const prompt = buildAIPrompt({
     profile,
     raceGoal,
     periodization,
     weeksUntilRace
   })

5. Chamada à API da IA
   ↓
   const response = await callOpenAI(prompt)

6. Parsing e Validação do JSON
   ↓
   const plan = JSON.parse(cleanMarkdown(response))
   validatePlanStructure(plan)

7. Criação de Workouts no Banco
   ↓
   for each week in plan.weeks:
     for each workout in week.workouts:
       await prisma.workout.create({...})

8. Retorno do Plano Completo
   ↓
   return {
     plan,
     rationale: plan.rationale,
     totalWeeks: weeksUntilRace
   }
```

Estrutura do Prompt de IA

```
const buildAIPrompt = (data) => `
Você é um treinador especialista em corrida que segue a metodologia VDOT de Jack
Daniels.

PERFIL DO ATLETA:
- Nome: ${profile.user.name}
- Idade: ${profile.age} anos
- Nível: ${profile.experienceLevel}
- VDOT Atual: ${profile.currentVDOT || 'A calcular'}
- Quilometragem Semanal Atual: ${profile.weeklyKm}km
- Dias disponíveis para treino: ${profile.availableDays.join(', ')}
- Dias para treino de força: ${profile.strengthDays.join(', ')}
- Histórico de lesões: ${profile.hasInjuries ? profile.injuryDetails : 'Nenhum'}

META DA PROVA:
- Prova: ${raceGoal.raceName}
- Data: ${raceGoal.raceDate}
- Distância: ${raceGoal.distance}km (${raceGoal.raceType})
- Objetivo: ${raceGoal.targetTime ? 'Tempo alvo: ' + raceGoal.targetTime : 'Completar
a prova'}
- Semanas até a prova: ${weeksUntilRace}

PERIODIZAÇÃO SUGERIDA:
- Fase Base: ${periodization.base} semanas
- Fase Build: ${periodization.build} semanas
- Fase Peak: ${periodization.peak} semanas
- Taper: ${periodization.taper} semanas

INSTRUÇÕES:
1. Gere um plano de ${weeksUntilRace} semanas
2. RESPEITE RIGOROSAMENTE os dias disponíveis: ${profile.availableDays.join(', ')}
3. Inclua treinos de força nos dias: ${profile.strengthDays.join(', ')}
4. Progressão gradual de volume (máximo 10% por semana)
5. Variedade de treinos:
   - Longões (long_run): 1x por semana
   - Intervalados (intervals): 1-2x por semana nas fases build/peak
   - Tempo runs: 1x por semana
   - Regenerativos (easy_run): para completar volume
   - Força (strength): nos dias especificados
   - Descanso (rest): 1-2x por semana

6. Use ritmos baseados em VDOT:
   - Easy: VDOT + 60-90s
   - Marathon Pace: VDOT + 30-45s
   - Threshold/Tempo: VDOT + 15-20s
   - Interval: VDOT pace
   - Repetition: VDOT - 10-15s

FORMATO DE SAÍDA (JSON PURO, SEM MARKDOWN):
{
  "plan": {
    "weeks": [
      {
        "weekNumber": 1,
        "phase": "base",
        "totalDistance": 45.0,
        "workouts": [
          {
            "day": "monday",
            "type": "easy_run",
            "description": "Corrida leve de recuperação",
            "distance": 8.0,
```

```
            "duration": 48,
            "pace": "5:30",
            "intensity": "Zone 2 (conversational)",
            "notes": "Foque em manter o ritmo confortável"
          },
          {
            "day": "tuesday",
            "type": "strength",
            "description": "Treino de força para corredores",
            "exercises": "Agachamento, Afundo, Prancha, Ponte Glúteos",
            "sets": "3x12",
            "duration": 45,
            "notes": "Priorize forma sobre carga"
          }
        ]
      }
    ]
  },
  "rationale": "Explicação detalhada da estratégia do plano, incluindo o porquê da
periodização escolhida e como o plano se adapta ao perfil do atleta."
}

IMPORTANTE:
- NÃO use dias que não estão em availableDays
- SEMPRE inclua força nos strengthDays
- Limite força a 4x por semana no máximo
- Retorne apenas JSON válido, sem ```json nem comentários
`;
```

Função de Periodização

```
function calculatePeriodization(totalWeeks: number, level: string) {
  // Percentuais por nível
  const ratios = {
    iniciante: { base: 0.50, build: 0.30, peak: 0.10, taper: 0.10 },
    intermediario: { base: 0.40, build: 0.35, peak: 0.15, taper: 0.10 },
    avancado: { base: 0.35, build: 0.35, peak: 0.20, taper: 0.10 }
  };

  const ratio = ratios[level] || ratios.intermediario;

  return {
    base: Math.ceil(totalWeeks * ratio.base),
    build: Math.ceil(totalWeeks * ratio.build),
    peak: Math.ceil(totalWeeks * ratio.peak),
    taper: Math.ceil(totalWeeks * ratio.taper)
  };
}
```

Tratamento de Erros e Retry

```
async function callOpenAIWithRetry(prompt: string, maxRetries = 3) {
  for (let attempt = 1; attempt <= maxRetries; attempt++) {
    try {
      const response = await fetch('https://api.abacus.ai/v1/chat/complete', {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json',
          'Authorization': `Bearer ${process.env.ABACUSAI_API_KEY}`
        },
        body: JSON.stringify({
          model: 'gpt-4o',
          messages: [
            { role: 'system', content: 'Você é um treinador expert de corrida.' },
            { role: 'user', content: prompt }
          ],
          temperature: 0.3,  // Baixa para consistência
          max_tokens: 8000
        })
      });

      const data = await response.json();
      const content = data.choices[0].message.content;

      // Limpa markdown se presente
      const cleaned = content.replace(/```json\n?/g, '').replace(/```\n?/g, '').trim()
;

      // Tenta parsear
      const parsed = JSON.parse(cleaned);

      // Valida estrutura básica
      if (!parsed.plan || !parsed.plan.weeks || !Array.isArray(parsed.plan.weeks)) {
        throw new Error('Estrutura de plano inválida');
      }

      console.log(`✅ Plano gerado com sucesso (tentativa ${attempt})`);
      return parsed;

    } catch (error) {
      console.error(`❌ Tentativa ${attempt} falhou:`, error.message);

      if (attempt === maxRetries) {
        throw new Error(`Falha ao gerar plano após ${maxRetries} tentativas`);
      }

      // Aguarda antes de retry (exponential backoff)
      await new Promise(resolve => setTimeout(resolve, 1000 * attempt));
    }
  }
}
```

🔗 Integração Strava

lib/strava.ts - Cliente Strava

OAuth Flow

```
// 1. Iniciar autorização
export function getStravaAuthUrl(userId: string) {
  const params = new URLSearchParams({
    client_id: process.env.STRAVA_CLIENT_ID!,
    redirect_uri: process.env.STRAVA_REDIRECT_URI!,
    response_type: 'code',
    scope: 'read,activity:read_all,profile:read_all',
    state: userId  // Para associar ao usuário correto
  });

  return `https://www.strava.com/oauth/authorize?${params}`;
}

// 2. Trocar code por tokens
export async function exchangeCodeForTokens(code: string) {
  const response = await fetch('https://www.strava.com/oauth/token', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      client_id: process.env.STRAVA_CLIENT_ID,
      client_secret: process.env.STRAVA_CLIENT_SECRET,
      code,
      grant_type: 'authorization_code'
    })
  });

  const data = await response.json();

  return {
    accessToken: data.access_token,
    refreshToken: data.refresh_token,
    expiresAt: data.expires_at,
    athleteId: data.athlete.id.toString()
  };
}

// 3. Salvar no banco
export async function saveStravaConnection(userId: string, tokens: Tokens) {
  return await prisma.stravaConnection.upsert({
    where: { userId },
    update: {
      accessToken: tokens.accessToken,
      refreshToken: tokens.refreshToken,
      expiresAt: tokens.expiresAt,
      athleteId: tokens.athleteId
    },
    create: {
      userId,
      ...tokens
    }
  });
}
```

Refresh Token Automático

```
export async function getValidAccessToken(userId: string) {
  const connection = await prisma.stravaConnection.findUnique({
    where: { userId }
  });

  if (!connection) {
    throw new Error('Strava não conectado');
  }

  // Verifica se token ainda é válido
  const now = Math.floor(Date.now() / 1000);
  if (connection.expiresAt > now + 300) {  // 5 min de margem
    return connection.accessToken;
  }

  // Token expirado, renova
  console.log('🔄 Renovando token Strava...');

  const response = await fetch('https://www.strava.com/oauth/token', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({
      client_id: process.env.STRAVA_CLIENT_ID,
      client_secret: process.env.STRAVA_CLIENT_SECRET,
      refresh_token: connection.refreshToken,
      grant_type: 'refresh_token'
    })
  });

  const data = await response.json();

  // Atualiza no banco
  await prisma.stravaConnection.update({
    where: { userId },
    data: {
      accessToken: data.access_token,
      refreshToken: data.refresh_token,
      expiresAt: data.expires_at
    }
  });

  return data.access_token;
}
```

Buscar Atividades

```
export async function getRecentActivities(
  userId: string,
  options: {
    startDate?: Date,
    endDate?: Date,
    page?: number,
    perPage?: number
  } = {}
) {
  const accessToken = await getValidAccessToken(userId);

  const params = new URLSearchParams({
    page: (options.page || 1).toString(),
    per_page: (options.perPage || 30).toString()
  });

  if (options.startDate) {
    params.append('after', Math.floor(options.startDate.getTime() / 1000).toString());
  }

  if (options.endDate) {
    params.append('before', Math.floor(options.endDate.getTime() / 1000).toString());
  }

  const response = await fetch(
    `https://www.strava.com/api/v3/athlete/activities?${params}`,
    {
      headers: {
        'Authorization': `Bearer ${accessToken}`
      }
    }
  );

  if (!response.ok) {
    throw new Error(`Strava API error: ${response.statusText}`);
  }

  return await response.json();
}
```

Calcular VDOT de Atividades

```
import { calculateVDOTFromPace } from './vdotTables';

export async function updateVDOTFromStrava(userId: string) {
  const activities = await getRecentActivities(userId, {
    startDate: new Date(Date.now() - 90 * 24 * 60 * 60 * 1000)  // 90 dias
  });

  // Filtra apenas corridas
  const runs = activities.filter(a => a.type === 'Run' && a.distance > 5000);

  if (runs.length === 0) {
    return null;
  }

  // Calcula VDOT para cada corrida
  const vdots = runs.map(run => {
    const distanceKm = run.distance / 1000;
    const timeMinutes = run.moving_time / 60;
    const paceMinPerKm = timeMinutes / distanceKm;

    return calculateVDOTFromPace(distanceKm, paceMinPerKm);
  }).filter(v => v > 0);

  if (vdots.length === 0) {
    return null;
  }

  // Pega a média dos melhores 20% (performances recentes)
  vdots.sort((a, b) => b - a);
  const top20Percent = vdots.slice(0, Math.max(1, Math.ceil(vdots.length * 0.2)));
  const avgVDOT = top20Percent.reduce((sum, v) => sum + v, 0) / top20Percent.length;

  // Atualiza no perfil
  await prisma.athleteProfile.update({
    where: { userId },
    data: { currentVDOT: Math.round(avgVDOT * 10) / 10 }
  });

  return avgVDOT;
}
```

Sincronizar Workouts Completos

```
export async function syncWorkoutsWithStrava(userId: string) {
  const activities = await getRecentActivities(userId, {
    startDate: new Date(Date.now() - 7 * 24 * 60 * 60 * 1000)  // última semana
  });

  const activePlan = await prisma.trainingPlan.findFirst({
    where: { userId, isActive: true },
    include: { workouts: true }
  });

  if (!activePlan) {
    return { synced: 0, message: 'Nenhum plano ativo' };
  }

  let syncedCount = 0;

  for (const activity of activities) {
    if (activity.type !== 'Run') continue;

    const activityDate = new Date(activity.start_date);

    // Encontra workout correspondente
    const matchingWorkout = activePlan.workouts.find(w => {
      const workoutDate = new Date(w.date);
      const sameDay =
        workoutDate.getFullYear() === activityDate.getFullYear() &&
        workoutDate.getMonth() === activityDate.getMonth() &&
        workoutDate.getDate() === activityDate.getDate();

      return sameDay && !w.completed;
    });

    if (matchingWorkout) {
      await prisma.workout.update({
        where: { id: matchingWorkout.id },
        data: {
          completed: true,
          completedAt: activityDate
        }
      });

      // Cria log do treino
      await prisma.workoutLog.create({
        data: {
          userId,
          workoutId: matchingWorkout.id,
          distance: activity.distance / 1000,
          duration: Math.floor(activity.moving_time / 60),
          pace: ((activity.moving_time / 60) / (activity.distance / 1000)).toFixed(2),
          feel: 5,  // neutro
          notes: `Sincronizado do Strava: ${activity.name}`,
          stravaActivityId: activity.id.toString()
        }
      });

      syncedCount++;
    }
  }

  return { synced: syncedCount, message: `${syncedCount} treinos sincronizados` };
}
```

🤖 Sistema de IA

Arquitetura de Chamadas

```
// lib/ai-client.ts

import OpenAI from 'openai';

const client = new OpenAI({
  apiKey: process.env.ABACUSAI_API_KEY,
  baseURL: 'https://api.abacus.ai/v1'
});

export async function callAI(
  messages: Array<{ role: string; content: string }>,
  options: {
    model?: string;
    temperature?: number;
    maxTokens?: number;
  } = {}
) {
  const response = await client.chat.completions.create({
    model: options.model || 'gpt-4o',
    messages,
    temperature: options.temperature ?? 0.3,
    max_tokens: options.maxTokens || 4000
  });

  return response.choices[0].message.content;
}
```

Chat de Treinamento

```
// /api/ai/chat/route.ts

export async function POST(req: Request) {
  const session = await getServerSession(authOptions);
  if (!session) {
    return NextResponse.json({ error: 'Não autenticado' }, { status: 401 });
  }

  const { message, context } = await req.json();

  // Busca contexto do usuário
  const profile = await prisma.athleteProfile.findUnique({
    where: { userId: session.user.id },
    include: { user: true }
  });

  const activePlan = await prisma.trainingPlan.findFirst({
    where: { userId: session.user.id, isActive: true },
    include: { workouts: true, raceGoal: true }
  });

  // Monta contexto para IA
  const systemPrompt = `
Você é um treinador pessoal especialista em corrida de rua.

Perfil do Atleta:
- Nome: ${profile?.user.name}
- Nível: ${profile?.experienceLevel}
- VDOT: ${profile?.currentVDOT || 'não calculado'}
- Objetivo: ${activePlan?.raceGoal.raceName} (${activePlan?.raceGoal.distance}km)

Responda de forma clara, motivadora e técnica. Use terminologia de corrida quando
apropriado, mas explique conceitos complexos.
`;

  const userMessage = message;

  const response = await callAI([
    { role: 'system', content: systemPrompt },
    { role: 'user', content: userMessage }
  ]);

  return NextResponse.json({ response });
}
```

Análise de Progresso

```
// /api/ai/analyze/route.ts

export async function POST(req: Request) {
  const session = await getServerSession(authOptions);
  if (!session) {
    return NextResponse.json({ error: 'Não autenticado' }, { status: 401 });
  }

  const userId = session.user.id;

  // Busca dados
  const logs = await prisma.workoutLog.findMany({
    where: { userId },
    orderBy: { createdAt: 'desc' },
    take: 30,
    include: { workout: true }
  });

  const profile = await prisma.athleteProfile.findUnique({
    where: { userId }
  });

  // Análise de consistência
  const completionRate = logs.filter(l => l.workout.completed).length / logs.length;
  const avgFeel = logs.reduce((sum, l) => sum + (l.feel || 5), 0) / logs.length;
  const avgPace = logs.reduce((sum, l) => sum + parseFloat(l.pace), 0) / logs.length;

  // Prompt para IA
  const analysisPrompt = `
Analise o progresso deste atleta:

Perfil:
- Nível: ${profile?.experienceLevel}
- VDOT: ${profile?.currentVDOT}

Estatísticas últimos 30 treinos:
- Taxa de conclusão: ${(completionRate * 100).toFixed(1)}%
- Sensação média: ${avgFeel.toFixed(1)}/10
- Pace médio: ${avgPace.toFixed(2)} min/km

Treinos recentes:
${logs.slice(0, 10).map(l =>
  `- ${l.workout.workoutType}: ${l.distance}km em ${l.duration}min (sensação $
{l.feel}/10)`
).join('\n')}

Forneça:
1. Análise do progresso
2. Pontos fortes
3. Áreas de melhoria
4. Recomendações específicas
5. Ajustes sugeridos (se necessário)

Formato JSON:
{
  "overall": "texto da análise geral",
  "strengths": ["ponto 1", "ponto 2"],
  "improvements": ["área 1", "área 2"],
  "recommendations": ["rec 1", "rec 2"],
  "adjustments": {
    "volume": "manter/aumentar/reduzir",
    "intensity": "manter/aumentar/reduzir",
```

```
    "reasoning": "explicação"
  }
}
`;

  const response = await callAI([
    { role: 'system', content: 'Você é um analista de performance de corrida.' },
    { role: 'user', content: analysisPrompt }
  ], {
    temperature: 0.4
  });

  const analysis = JSON.parse(response.replace(/```json\n?/g, '').replace(/```/g, ''))
;

  return NextResponse.json({ analysis });
}
```

💽 Banco de Dados

Comandos Prisma Essenciais

```
# Gerar Prisma Client após alterar schema
yarn prisma generate

# Sincronizar schema com banco (dev)
yarn prisma db push

# Criar migration (produção)
yarn prisma migrate dev --name nome_da_migration

# Visualizar banco em GUI
yarn prisma studio
# Acessa http://localhost:5555

# Resetar banco (CUIDADO: apaga tudo!)
yarn prisma migrate reset

# Seed do banco
yarn prisma db seed
```

Queries Comuns

```
// Buscar perfil completo do usuário
const fullProfile = await prisma.user.findUnique({
  where: { email: 'user@example.com' },
  include: {
    profile: true,
    plans: {
      where: { isActive: true },
      include: {
        workouts: {
          where: { completed: false },
          orderBy: { date: 'asc' }
        },
        raceGoal: true
      }
    },
    raceGoals: {
      orderBy: { raceDate: 'asc' }
    },
    stravaConnection: true
  }
});

// Buscar treinos da semana
const startOfWeek = new Date();
startOfWeek.setHours(0, 0, 0, 0);
startOfWeek.setDate(startOfWeek.getDate() - startOfWeek.getDay() + 1); // segunda

const endOfWeek = new Date(startOfWeek);
endOfWeek.setDate(endOfWeek.getDate() + 6); // domingo

const weekWorkouts = await prisma.workout.findMany({
  where: {
    plan: { userId },
    date: {
      gte: startOfWeek,
      lte: endOfWeek
    }
  },
  orderBy: { date: 'asc' },
  include: {
    logs: true
  }
});

// Estatísticas do usuário
const stats = await prisma.workoutLog.groupBy({
  by: ['userId'],
  where: {
    userId,
    createdAt: {
      gte: new Date(Date.now() - 30 * 24 * 60 * 60 * 1000) // últimos 30 dias
    }
  },
  _sum: {
    distance: true,
    duration: true
  },
  _count: {
    id: true
  },
  _avg: {
    feel: true
```

```
  }
});
```

Transações

```
// Criar plano e workouts atomicamente
const result = await prisma.$transaction(async (tx) => {
  // 1. Desativa planos anteriores
  await tx.trainingPlan.updateMany({
    where: { userId, isActive: true },
    data: { isActive: false }
  });

  // 2. Cria novo plano
  const plan = await tx.trainingPlan.create({
    data: {
      userId,
      raceGoalId,
      planType: 'intermediate',
      totalWeeks,
      startDate,
      endDate,
      basePhaseWeeks,
      buildPhaseWeeks,
      peakPhaseWeeks,
      taperWeeks,
      aiGeneratedData: generatedPlan,
      rationale: generatedPlan.rationale,
      isActive: true
    }
  });

  // 3. Cria todos os workouts
  const workouts = [];
  for (const week of generatedPlan.plan.weeks) {
    for (const workout of week.workouts) {
      workouts.push({
        planId: plan.id,
        weekNumber: week.weekNumber,
        dayOfWeek: workout.day,
        date: calculateDate(startDate, week.weekNumber, workout.day),
        workoutType: workout.type,
        description: workout.description,
        targetDistance: workout.distance,
        targetDuration: workout.duration,
        targetPace: workout.pace,
        targetIntensity: workout.intensity
      });
    }
  }

  await tx.workout.createMany({
    data: workouts
  });

  return plan;
});
```

🧪 Testes

Script de Teste Abrangente

Localização: `scripts/comprehensive_test.ts`

```
import { PrismaClient } from '@prisma/client';
import bcrypt from 'bcryptjs';
import { generateTrainingPlan } from '../lib/ai-plan-generator';

const prisma = new PrismaClient();

async function createTestUser(email: string, profile: any, raceGoal: any) {
  // Cria usuário
  const passwordHash = await bcrypt.hash('senha123', 10);
  const user = await prisma.user.create({
    data: {
      email,
      passwordHash,
      name: email.split('@')[0]
    }
  });

  // Cria perfil
  const athleteProfile = await prisma.athleteProfile.create({
    data: {
      userId: user.id,
      ...profile
    }
  });

  // Cria meta de prova
  const goal = await prisma.raceGoal.create({
    data: {
      userId: user.id,
      ...raceGoal
    }
  });

  return { user, athleteProfile, goal };
}

async function runTests() {
  console.log('🧪 Iniciando testes abrangentes...\n');

  const testCases = [
    {
      name: 'Iniciante - 10km',
      email: 'teste.iniciante@example.com',
      profile: {
        age: 28,
        gender: 'M',
        weight: 75,
        height: 175,
        experienceLevel: 'iniciante',
        weeklyKm: 15,
        trainingDaysPerWeek: 4,
        availableDays: ['monday', 'wednesday', 'friday', 'sunday'],
        strengthDays: ['tuesday'],
        currentVDOT: 35
      },
      raceGoal: {
        raceName: 'Corrida de Rua 10km',
        raceDate: new Date(Date.now() + 120 * 24 * 60 * 60 * 1000), // 120 dias
        distance: 10,
        raceType: '10k',
        targetTime: '00:55:00',
        isPrimary: true
```

```
      }
    },
    {
      name: 'Intermediário - Meia Maratona',
      email: 'teste.intermediario@example.com',
      profile: {
        age: 35,
        gender: 'F',
        weight: 60,
        height: 165,
        experienceLevel: 'intermediario',
        weeklyKm: 40,
        trainingDaysPerWeek: 5,
        availableDays: ['monday', 'tuesday', 'thursday', 'saturday', 'sunday'],
        strengthDays: ['wednesday', 'friday'],
        currentVDOT: 45
      },
      raceGoal: {
        raceName: 'Meia Maratona',
        raceDate: new Date(Date.now() + 180 * 24 * 60 * 60 * 1000), // 180 dias
        distance: 21.0975,
        raceType: 'half_marathon',
        targetTime: '01:45:00',
        isPrimary: true
      }
    },
    {
      name: 'Avançado - Maratona',
      email: 'teste.avancado@example.com',
      profile: {
        age: 32,
        gender: 'M',
        weight: 70,
        height: 180,
        experienceLevel: 'avancado',
        weeklyKm: 80,
        trainingDaysPerWeek: 6,
        availableDays: ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'sat-
urday'],
        strengthDays: ['sunday'],
        currentVDOT: 55
      },
      raceGoal: {
        raceName: 'Maratona de São Paulo',
        raceDate: new Date(Date.now() + 240 * 24 * 60 * 60 * 1000), // 240 dias
        distance: 42.195,
        raceType: 'marathon',
        targetTime: '03:15:00',
        isPrimary: true
      }
    }
  ];

  const results = [];

  for (const testCase of testCases) {
    console.log(`\n${'='.repeat(60)}`);
    console.log(`📋 Testando: ${testCase.name}`);
    console.log(`${'='.repeat(60)}\n`);

    try {
      // Limpa usuário de teste anterior
      await prisma.user.deleteMany({
```

```
        where: { email: testCase.email }
      });

      // Cria usuário e perfil
      const { user, athleteProfile, goal } = await createTestUser(
        testCase.email,
        testCase.profile,
        testCase.raceGoal
      );

      console.log(`✅ Usuário criado: ${user.email}`);
      console.log(`✅ Perfil criado: ${athleteProfile.experienceLevel}, ${athletePro-
file.trainingDaysPerWeek} dias/semana`);
      console.log(`✅ Meta criada: ${goal.raceName} em ${goal.raceDate.toLocaleDateStr
ing()}`);

      // Gera plano
      console.log('\n⏳ Gerando plano de treino...');
      const plan = await generateTrainingPlan(athleteProfile, goal);

      console.log(`\n✅ Plano gerado com sucesso!`);
      console.log(`   - Duração: ${plan.totalWeeks} semanas`);
      console.log(`   - Total de treinos: ${plan.plan.weeks.reduce((sum, w) => sum + w
.workouts.length, 0)}`);
      console.log(`   - Periodização: Base=${plan.basePhaseWeeks}, Build=${plan.buildP
haseWeeks}, Peak=${plan.peakPhaseWeeks}, Taper=${plan.taperWeeks}`);

      // Valida disponibilidade
      const allWorkouts = plan.plan.weeks.flatMap(w => w.workouts);
      const invalidDays = allWorkouts.filter(w =>
        !testCase.profile.availableDays.includes(w.day) &&
        !testCase.profile.strengthDays.includes(w.day)
      );

      if (invalidDays.length > 0) {
        console.warn(`   ⚠️ ${invalidDays.length} treinos em dias não disponíveis!`);
        results.push({ testCase: testCase.name, success: false, reason: 'Dias in-
válidos' });
      } else {
        console.log(`   ✅ Todos os treinos respeitam disponibilidade`);
      }

      // Valida força
      const strengthWorkouts = allWorkouts.filter(w => w.type === 'strength');
      const strengthOnCorrectDays = strengthWorkouts.filter(w =>
        testCase.profile.strengthDays.includes(w.day)
      );

      console.log(`   ✅ Treinos de força: ${strengthWorkouts.length} (${strengthOn-
CorrectDays.length} nos dias corretos)`);

      results.push({ testCase: testCase.name, success: true, plan });

    } catch (error) {
      console.error(`❌ ERRO: ${error.message}`);
      results.push({ testCase: testCase.name, success: false, error: error.message });
    }
  }

  // Resumo
  console.log(`\n\n${'='.repeat(60)}`);
  console.log('📊 RESUMO DOS TESTES');
  console.log(`${'='.repeat(60)}\n`);
```

```
  const successCount = results.filter(r => r.success).length;
  const totalCount = results.length;

  results.forEach(r => {
    console.log(`${r.success ? '✅' : '❌'} ${r.testCase}`);
    if (!r.success && r.error) {
      console.log(`   Erro: ${r.error}`);
    }
  });

  console.log(`\n🎯 Taxa de Sucesso: ${successCount}/${totalCount} (${((successCount/t
otalCount) * 100).toFixed(1)}%)`);

  await prisma.$disconnect();
}

runTests().catch(console.error);
```

Executar Testes

```
# Teste abrangente
yarn ts-node scripts/comprehensive_test.ts

# Teste específico de geração
yarn ts-node test_plan_generation.ts

# Teste de usuário específico
yarn ts-node scripts/check_user.ts

# Teste de perfil
yarn ts-node check_profile_data.ts
```

🚀 Deploy

Build de Produção

```
# 1. Instala dependências
yarn install --frozen-lockfile

# 2. Gera Prisma Client
yarn prisma generate

# 3. Roda migrations
yarn prisma migrate deploy

# 4. Build do Next.js
yarn build

# 5. Inicia servidor
yarn start
```

Variáveis de Ambiente em Produção

```
# Certifique-se de que todas as variáveis estão definidas
DATABASE_URL="postgresql://..."
NEXTAUTH_SECRET="strong-secret-key"
NEXTAUTH_URL="https://42maurillio.abacusai.app"
STRAVA_CLIENT_ID="..."
STRAVA_CLIENT_SECRET="..."
STRAVA_REDIRECT_URI="https://42maurillio.abacusai.app/api/strava/callback"
ABACUSAI_API_KEY="..."
NODE_ENV="production"
```

Checklist de Deploy

- [ ] Variáveis de ambiente configuradas
- [ ] Migrations rodadas
- [ ] Prisma Client gerado
- [ ] Build passa sem erros
- [ ] Testes passam
- [ ] Callback do Strava atualizado
- [ ] NEXTAUTH_URL aponta para domínio correto
- [ ] Banco de dados acessível

🔧 Troubleshooting

Problema: “Prisma Client not generated”

```
# Solução
yarn prisma generate
```

Problema: “Cannot connect to database”

```
# Verificar conexão
yarn prisma db push

# Se falhar, verifique DATABASE_URL em .env
```

Problema: “NextAuth session not found”

```
// Verificar se middleware está configurado
// middleware.ts
export { default } from "next-auth/middleware";

export const config = {
  matcher: ['/dashboard/:path*', '/plano/:path*', '/perfil/:path*']
};
```

Problema: “Strava OAuth fails”

1. Verifique se `STRAVA_REDIRECT_URI` está correto

2. Verifique se callback está registrado no Strava App
3. Verifique se escopos estão corretos: `read,activity:read_all`

Problema: “AI generates invalid JSON”

- Verifique `ABACUSAI_API_KEY`
- Sistema tem retry automático
- Verifique logs para ver resposta da IA
- Considere aumentar `maxRetries` em `ai-plan-generator.ts`

Problema: “Workouts on wrong days”

1. Verifique se `availableDays` está salvo corretamente no perfil
2. Verifique se `strengthDays` está salvo
3. Rode testes: `yarn ts-node scripts/comprehensive_test.ts`
4. Veja logs da geração do plano

Logs de Debug

```
// Adicionar em qualquer lugar para debug
console.log('🔍 Debug:', JSON.stringify(data, null, 2));

// Para APIs
console.log('📨 Request:', req.body);
console.log('📤 Response:', response);

// Para IA
console.log('🤖 AI Prompt:', prompt);
console.log('🤖 AI Response:', aiResponse);
```

📚 Recursos Adicionais

Documentação Externa

- Next.js Docs (https://nextjs.org/docs)
- Prisma Docs (https://www.prisma.io/docs)
- NextAuth.js (https://next-auth.js.org)
- Strava API (https://developers.strava.com)
- Shadcn UI (https://ui.shadcn.com)
- Jack Daniels VDOT (https://runsmartproject.com/calculator/)

Comandos Úteis

```
# Ver estrutura do banco
yarn prisma studio

# Formatar código
yarn prettier --write .

# Lint
yarn next lint

# Tipo check
yarn tsc --noEmit

# Ver build size
yarn next build --profile

# Analisar bundle
yarn next build --analyze
```

Última Atualização: 27 de outubro de 2025
Versão: 1.0.0
Mantenedor: Maurillio (mmaurillio2@gmail.com)