Resumo do projeto

Maratona Tech — plataforma web para múltiplos eventos (hackathons/maratonas) onde:

- administradores criam eventos e pitchs;
- usuários (público) se cadastram e votam (1 voto por usuário por evento);
- jurados autenticados avaliam tecnicamente (notas por critérios);
- exibe rankings: Popular, Técnico e Combinado (50% Popular + 50% Técnico);
- landing page personalizável pelo admin que pode exibir pitchs de um evento (apenas quando votação aberta).

Stack recomendada: **Django + Django REST Framework (DRF)**; DB: **PostgreSQL**; Cache/Tasks: **Redis + Celery**; Websockets (opcional): **Django Channels**; Storage: **S3 (ou equivalente)**; Deploy: **Gunicorn + Nginx**.

Entregáveis esperados ao final

1. Backend Django (API REST) com autenticação (JWT or session), autorização, validações e testes.
2. Painel admin customizado (Django Admin) para eventos/pitchs/usuários/jurados.
3. Frontend (pode ser separado) ou endpoints prontos para consumir por um app no-code.
4. Documentação da API (OpenAPI/Swagger).
5. Scripts de migração e instruções de deploy.
6. Testes unitários + integração (voto, cálculo de média, regras de permissão).
7. Observabilidade mínima (logs, métricas básicas, alertas).

Regras de negócio (detalhadas)

- **Eventos**
    - Um evento tem: título, descrição, banner, data_inicio_votacao, data_fim_votacao, ativo(bool), exibir_na_home(bool).
    - **Apenas 1 evento** pode ter `exibir_na_home = True` ao mesmo tempo (validar no admin/server).
    - Eventos sempre públicos (visíveis), votos apenas no período definido.
- **Pitch**
    - Cada pitch pertence a um evento.
    - Campos: título, descrição, galeria (imagens), vídeo (url), integrantes (texto), tags, total_votos (int), media_tecnica (float), pontuacao_combinada (float).
    - Pitchs são visíveis publicamente, mas só recebem votos quando o evento está ativo.
- **Usuários**
    - Papéis: ADMIN, JURADO, VOTANTE (usuário comum).
    - Cadastro necessário para votar. Jurados são marcados com flag/role.
- **Votação Popular**
    - Cada usuário autenticado pode votar **1 vez por evento**.
    - Cada voto adiciona +1 ao `Pitch.total_votos`.
    - Antes de registrar o voto, exibir confirmação: *"Deseja confirmar o voto em (Nome do Projeto)?"*
    - Evitar duplo envio: botão desabilitado + verificação/insert atômico server-side + unique constraint (usuario, evento) na tabela de votos.
- **Avaliação Técnica (Jurados)**
    - Jurado pode avaliar cada pitch **uma vez por evento**.
    - Notas: Inovação (0–10), Design (0–10), Aplicabilidade (0–10). Média individual = média(3 notas).
    - `pitch.media_tecnica` = AVG(media_individual_jurados).
    - Notas técnicas não alteram contagem de votos populares.

    - Ranking técnico: ordenar por `media_tecnica` desc.

- **Cálculo Combinado**
    - Para cada pitch no evento:
        - `popular_percent = (pitch.total_votos / max_total_votos_no_evento) * 100` (se max == 0 → consider 0)
        - `tecnico_percent = (pitch.media_tecnica / 10) * 100` (porque média técnica 0–10)
        - `pontuacao_combinada = 0.5 * popular_percent + 0.5 * tecnico_percent`
    - Ordenar ranking combinado por `pontuacao_combinada` desc.
    - Exibir com 2 decimais.
- **Landing Page**
    - Admin edita banner + texto.
    - Se existe evento com `exibir_na_home=True` e evento.status == active → exibir pitchs desse evento na home (votáveis, se login).
    - Se não, listar eventos com seu status.
- **Logs/Auditoria**
    - Registrar logs de voto: usuário, pitch, evento, timestamp, ip, user_agent.
    - Registrar logs de avaliação técnica similar.

Modelos Django (sugestão) — versões resumidas

```
# apps/core/models.py
from django.db import models
from django.conf import settings
from django.utils import timezone
```

```
class ClinicEvent(models.Model):
    title = models.CharField(max_length=255)
    description = models.TextField(blank=True)
    banner = models.ImageField(upload_to="events/banners/", blank=True,
null=True)
    logo = models.ImageField(upload_to="events/logos/", blank=True,
null=True)
    voting_start = models.DateTimeField()
    voting_end = models.DateTimeField()
    active = models.BooleanField(default=True)
    show_on_home = models.BooleanField(default=False)

    class Meta:
        ordering = ['-voting_start']

    @property
    def status(self):
        now = timezone.now()
        if now < self.voting_start:
            return "Aguardando Início"
        if self.voting_start <= now <= self.voting_end:
            return "Votação Ativa"
        return "Encerrada"

class Pitch(models.Model):
    event = models.ForeignKey(ClinicEvent, related_name='pitches',
on_delete=models.CASCADE)
    title = models.CharField(max_length=255)
    description = models.TextField()
    gallery = models.JSONField(default=list)  # or a related Image model
    video_url = models.URLField(blank=True, null=True)
    team_members = models.TextField(blank=True)
    tags = models.JSONField(default=list)
    total_votes = models.PositiveIntegerField(default=0)
    media_tecnica = models.FloatField(default=0.0)
    pontuacao_combinada = models.FloatField(default=0.0)

    class Meta:
        unique_together = ('event', 'title')

# Voto & Avaliacao
class Vote(models.Model):
```

```
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE)
    pitch = models.ForeignKey(Pitch, on_delete=models.CASCADE,
related_name='votes')
    event = models.ForeignKey(ClinicEvent, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    ip_address = models.GenericIPAddressField(null=True, blank=True)
    user_agent = models.TextField(null=True, blank=True)

    class Meta:
        unique_together = ('user', 'event')

class TechnicalEvaluation(models.Model):
    juror = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, limit_choices_to={'is_juror': True})
    pitch = models.ForeignKey(Pitch, on_delete=models.CASCADE,
related_name='evaluations')
    event = models.ForeignKey(ClinicEvent, on_delete=models.CASCADE)
    inovacao = models.DecimalField(max_digits=4, decimal_places=2)
    design = models.DecimalField(max_digits=4, decimal_places=2)
    aplicabilidade = models.DecimalField(max_digits=4, decimal_places=2)
    media_final = models.DecimalField(max_digits=5, decimal_places=2)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        unique_together = ('juror', 'pitch', 'event')
```

Notas:

- Ajuste tipos (Decimal vs Float) conforme política financeira/precisão.
- Adicionar índices: `Vote(user,event)` unique index; index on `Pitch.event, total_votes` for ranking.

Fluxos críticos e snippets (ex.: registrar voto atômico)

Objetivo: evitar votos duplicados e race conditions. Implementar com transação/SELECT ... FOR UPDATE or DB constraint and catching IntegrityError.

```
from django.db import transaction, IntegrityError
from django.utils import timezone
from django.http import JsonResponse
from rest_framework.decorators import api_view, permission_classes
from rest_framework.permissions import IsAuthenticated

@api_view(['POST'])
@permission_classes([IsAuthenticated])
def vote_pitch(request, pitch_id):
    user = request.user
    try:
        pitch = Pitch.objects.select_related('event').get(pk=pitch_id)
    except Pitch.DoesNotExist:
        return JsonResponse({'detail': 'Pitch not found'}, status=404)

    event = pitch.event
    now = timezone.now()
    if not (event.voting_start <= now <= event.voting_end):
        return JsonResponse({'detail': 'Votação não ativa'}, status=400)

    # server-side atomic check + insert
    try:
        with transaction.atomic():
            # Create vote (unique constraint on user+event in Vote model)
            vote = Vote.objects.create(user=user, pitch=pitch,
event=event,
                                       ip_address=get_client_ip(request),

user_agent=request.META.get('HTTP_USER_AGENT', ''))
            # Update pitch total atomically

Pitch.objects.filter(pk=pitch.pk).update(total_votes=F('total_votes') +
1)
    except IntegrityError:
        return JsonResponse({'detail': 'Você já votou neste evento.'},
status=400)

    # Optionally enqueue task to recompute combined score or update cache
    recompute_pontuacao_for_event.delay(event.id)

    return JsonResponse({'detail': 'Voto registrado', 'total_votes':
pitch.total_votes + 1})
```

Observações:

- `unique_together = ('user', 'event')` na model Vote garante integridade.
- Catch `IntegrityError` para responder quando voto duplicado.
- Use `F()` para incremento atômico.

Cálculo de pontuação combinada (recomenda-se como tarefa assíncrona)

- Recompute para event: buscar `max_total_votos = Pitch.objects.filter(event=event).aggregate(Max('total_votes'))['total']`
- For each pitch:
  - `popular_percent = (pitch.total_votes / (max_total_votos or 1)) * 100`
  - `tecnico_percent = (pitch.media_tecnica / 10) * 100`
  - `pontuacao_combinada = 0.5 * popular_percent + 0.5 * tecnico_percent`
  - Salvar `pitch.pontuacao_combinada`

Implementar isso em Celery task `recompute_pontuacao_for_event`.

API endpoints (DRF style) — resumo

Authentication:

- `POST /api/auth/login/` → retorna JWT
- `POST /api/auth/register/` → registro

- `POST /api/auth/logout/`

Events:

- `GET /api/events/` → listar (público)
- `GET /api/events/{id}/` → detalhes
- `POST /api/events/` → admin only
- `PATCH /api/events/{id}/` → admin only

Pitches:

- `GET /api/events/{event_id}/pitches/`
- `GET /api/pitches/{id}/`
- `POST /api/events/{event_id}/pitches/` → admin
- `PATCH /api/pitches/{id}/` → admin

Voting:

- `POST /api/pitches/{id}/vote/` → authenticated user (see server atomic check)
- `GET /api/events/{id}/votes/` → admin (metrics)

Technical evaluation:

- `POST /api/pitches/{id}/evaluate/` → juror only
- `GET /api/pitches/{id}/evaluations/` → admin/juror

Ranking:

- `GET /api/events/{id}/ranking/popular/`
- `GET /api/events/{id}/ranking/technical/`
- `GET /api/events/{id}/ranking/combined/`

Dashboard:

- `GET /api/admin/dashboard/metrics/` → admin only

Landing page:

- `GET /api/home/` → returns banner, text, and pitches/events depending on show_on_home

Files:

- `POST /api/uploads/` → signed URL or direct upload to S3 (recommended)

Requisitos funcionais (explicados para o dev)

1. Autenticação segura (preferível JWT via `djangorestframework-simplejwt` ou sessions + CSRF para sites monolíticos).
2. Permissões por papel (Django groups/roles): ADMIN, JUROR, VOTER.
3. CRUD completo para Eventos, Pitches, Usuários, Votos, Avaliações.
4. Página inicial editável: banner + texto (save in model `SiteConfiguration`).
5. Exibição condicional na home (campo `show_on_home`).
6. Votação: popup de confirmação no frontend; backend registra voto atomically; unique constraint user+event.
7. Avaliação técnica: jurado avalia com notas e observations; average calculation.
8. Rankings: endpoints que retornem listas ordenadas.
9. Admin dashboard: endpoints com metrics, e Django Admin custom views.
10. Export de rankings (CSV/PDF) — endpoint e admin action.
11. Logs de auditoria (votos/avaliações) com ip/ua/timestamp.

Requisitos não-funcionais

1. **Performance & Escalabilidade**
    - PostgreSQL; índices em `Vote(user,event)`, `Pitch(event,total_votes)`; use Redis para cache ranking.
    - Celery + Redis para tarefas demoradas (recompute rankings).
    - Web server: Gunicorn + Nginx; Horizontal scaling de app workers e Celery workers.
2. **Disponibilidade**
    - Backups diários do banco (pg_dump) e retenção 7–30 dias.
    - Healthcheck endpoint (e.g., `/health/`) para orquestradores.
3. **Observabilidade**
    - Logs estruturados (JSON), integração com Sentry.
    - Métricas (Prometheus/Grafana) para requests, erros, latência, tasks failures.
4. **Testabilidade**
    - Unit tests para models, API tests para endpoints críticos (voto, avaliação).
    - CI pipeline (GitHub Actions/GitLab CI) rodando tests e linters.
5. **UX**
    - API paginada, responses claras, mensagens de erro padronizadas.
    - Timeouts e retries controlados no frontend.

Requisitos de segurança (essenciais)

1. **Autenticação & Senhas**
    - Senhas com hashing (Django usa PBKDF2 por padrão) — considerar Argon2.
    - Política de senha (min length, complexity) e reset via e-mail com token temporário.

    - 2FA opcional para administradores e jurados (ex.: TOTP).

2. **Autorização**
    - Verificações de permissão em todas as views (DRF permission classes).
    - Nunca confiar no frontend para autorizar (server-side authoritative).
3. **Proteção de APIs**
    - Rate limiting (e.g. django-ratelimit / nginx) para proteção contra abuso (voto scraping).
    - CSRF proteção para endpoints com cookies; usar JWT for SPA.
4. **Integração de arquivos**
    - Uploads → armazenar em S3 com URL assinada; validar tipos MIME; tamanho máximo; escanear malware (opcional).
    - Não armazenar arquivos executáveis.
5. **Proteção contra ataques comuns**
    - SQL injection mitigado pelo ORM.
    - XSS: escapar conteúdo no frontend; sanitize rich text fields or store as plain text.
    - Clickjacking: `X-Frame-Options: DENY`.
    - HSTS, Content Security Policy (CSP), secure cookies (HttpOnly, Secure).
6. **Dados Pessoais / LGPD**
    - Minimizar dados pessoais armazenados; política de privacidade; opção de remoção/anonimização.
    - Registrar consentimento (booleans) se for necessário.
7. **Segurança no deploy**
    - Usar HTTPS (Let's Encrypt).
    - Segredos via environment variables / secrets manager.
    - Não commitar `settings.py` com secrets.
8. **Auditoria**

    - Registrar ações críticas (criar/editar/excluir eventos/pitchs, exportar ranking).
    - Logs de auditoria devem incluir usuário, timestamp, action e IP.

Testes essenciais que o dev deve entregar

- Unit tests:
    - Model validations (unique constraints, computed fields).
    - `Vote` uniqueness logic (try to create duplicate and assert IntegrityError).
    - Calculation function for pontuacao_combinada.
- Integration tests (API):
    - Voting flow: unauthenticated blocked, authenticated allowed once, second attempt blocked.
    - Evaluation flow: juror can submit, average updates.
    - Ranking endpoints return correct order.
- E2E (optional): simulate user and juror flows.

Observações de arquitetura e performance

- Use **database transactions** e **unique constraints** como fontes de verdade; não apenas verificações no app.
- Para contagem de votos em alto volume, considerar **counter table** or Redis counter with periodic persistence.
- Cache ranking responses and invalidate/refresh on events like new vote or periodic (short TTL).
- Para atualizações em tempo real (ranking), use **Django Channels** + WebSockets ou polling no frontend.

Deployment / DevOps checklist (mínimo)

- PostgreSQL (prod), Redis, Celery worker, Gunicorn app servers, Nginx reverse proxy.
- CI: tests + flake8/black + migrations check.
- CD: scripts de migração automática com rollback plan.
- Backups automatizados (DB + uploaded files).
- Environment variables: `SECRET_KEY`, DB creds, S3 keys, JWT secret, etc.
- HTTPS enabled; HSTS; secure cookie flags.

Documentação para o desenvolvedor (entregável a ele)

1. Arquivo `README.md` com:
    - Setup local (env vars, docker-compose example)
    - Migrations & runserver
    - How to run tests
2. API docs (Swagger/OpenAPI) via DRF schema + drf-yasg.
3. ER diagram (models + relations).
4. Checklist final de QA (testes manuais/automáticos).
5. Guideline para frontend: endpoints, payload examples.

Snippets úteis adicionais

Recompute média técnica (Celery task)

```
from celery import shared_task
from django.db.models import Avg, Max
```

```
from .models import Pitch

@shared_task
def recompute_pontuacao_for_event(event_id):
    pitches = Pitch.objects.filter(event_id=event_id)
    max_votes = pitches.aggregate(max_v=Max('total_votes'))['max_v'] or 0
    for p in pitches:
        popular_percent = (p.total_votes / max_votes)*100 if max_votes
else 0
        tecnico_percent = (p.media_tecnica / 10) * 100
        p.pontuacao_combinada = 0.5 * popular_percent + 0.5 *
tecnico_percent
        p.save(update_fields=['pontuacao_combinada'])
```

Admin validation: only one show_on_home

```
from django.core.exceptions import ValidationError

def clean(self):
    if self.show_on_home:
        if
ClinicEvent.objects.filter(show_on_home=True).exclude(pk=self.pk).exists(
):
            raise ValidationError("Já existe um evento marcado para
exibir na home.")
```

Checklist pronto para enviar ao dev (resumido)

- Implement models (Event, Pitch, Vote, TechnicalEvaluation, User roles)
- Migrations and DB constraints (unique_together on Vote (user,event) and Evaluation (juror,pitch,event))
- DRF serializers & viewsets + permission classes
- Atomic vote flow with unique constraint + IntegrityError handling
- Celery task to recompute pontuacao_combinada

- Endpoints for rankings and dashboard metrics
- File upload integration (S3 recommended) and validation
- Tests: unit, integration for critical flows
- Admin UI and export CSV/PDF
- Security hardening: HTTPS, env secrets, rate-limiting, input sanitization
- Observability: Sentry + basic metrics + log format
- CI pipeline and deploy scripts + backup strategy

📌 Visão Geral do Projeto

O **Maratona Tech** é uma plataforma web para gestão de eventos de pitch, permitindo cadastro de eventos, pitchs, usuários participantes e votação online. O foco principal é oferecer um ambiente seguro, simples e escalável para que os participantes votem no melhor pitch durante competições, hackathons e maratonas de tecnologia.

O sistema inclui:

- Frontend web responsivo
- Backend em Django
- Banco de dados relacional
- Painel administrativo resumido
- Regras de segurança e auditoria
- API para integrações futuras

🧩 Arquitetura Geral

- **Framework Backend:** Django 5+
- **Banco de Dados:** PostgreSQL
- **Autenticação:** Django Authentication + JWT para API (caso necessário)
- **Front-End:** Django Templates, TailwindCSS ou Bootstrap
- **Admin:** Django Admin + Painel Personalizado
- **Deploy:** Docker + Render/Heroku/AWS/Oracle Free Tier
- **Cache:** Redis (opcional para performance)

📌 Requisitos Funcionais (RF)

RF001 – Cadastro e Autenticação de Usuários

- RF001.1 – Usuários podem se cadastrar com email + senha.
- RF001.2 – Administradores criam contas especiais (admin/evento/visualizador).
- RF001.3 – Login com validação e recuperação de senha.

RF002 – Gestão de Eventos

- RF002.1 – Administradores podem criar eventos (nome, logo, descrição, data/hora).
- RF002.2 – Cada evento tem chave/código único para acesso dos participantes.
- RF002.3 – Eventos podem ser ativados/desativados.

RF003 – Gestão de Pitchs

- RF003.1 – Cadastro de pitchs vinculados a um evento.
- RF003.2 – Cada pitch contém: título, equipe, descrição, imagem/logo.
- RF003.3 – Administradores podem editar e excluir pitchs.

RF004 – Sistema de Votação

- RF004.1 – Usuários logados podem votar em um pitch por evento.
- RF004.2 – Apenas **1 voto por usuário por evento**.
- RF004.3 – Voto deve ser registrado imediatamente no banco de dados.
- RF004.4 – Sistema impede voto duplicado.

RF005 – Painel Administrativo Resumido (Real-Time)

- RF005.1 – Gráfico com número de votos por pitch.
- RF005.2 – Pitch mais votado em tempo real.
- RF005.3 – Número total de usuários ativos.
- RF005.4 – Total de votos contabilizados.
- RF005.5 – Exportação em CSV/Excel (opcional).

RF006 – Dashboard Público (Opcional)

- RF006.1 – Exibir ranking em tempo real sem mostrar quantidades exatas (opcional).
- RF006.2 – Página com QR Code do evento para votação.

RF007 – Log e Auditoria

- RF007.1 – Registrar ações importantes no sistema.
- RF007.2 – Registrar IP, user agent e horário do voto.

📌 Requisitos Não Funcionais (RNF)

RNF001 – Segurança

- Senhas hash com **PBKDF2** (padrão Django).
- Anti-CSRF nas rotas com formulários.
- Anti-XSS e Anti-SQL Injection (Django ORM).
- JWT deve expirar e usar refresh tokens se habilitado.
- Limitação de tentativas de login.
- Regras CORS configuradas se houver API.
- Criptografia TLS no deploy.

RNF002 – Performance

- Uso de cache em:
    - ranking de pitchs
    - lista de eventos
- Otimização de queries com `select_related` e `prefetch_related`.

RNF003 – Escalabilidade

- Sistema deve permitir:

- múltiplos eventos simultâneos
- milhares de votos/minuto

RNF004 – Usabilidade

- Design responsivo e mobile-first.
- Interface simples para usuários leigos.

RNF005 – Confiabilidade

- Registro de voto deve ser atômico (transações).
- Falhas devem retornar mensagens amigáveis.

🔐 Regras de Segurança Específicas

- Um usuário só pode votar **após login**.
- Um usuário não pode votar duas vezes no mesmo evento.
- Backend valida todos os votos independentemente do frontend.
- Votos só podem ser criados, nunca editados ou apagados (auditoria).
- Eventos inativos não aceitam novos votos.

🧠 Regras de Negócio

RN001 – Um voto por usuário por evento

- O mesmo usuário não pode votar em dois pitchs diferentes dentro do mesmo evento.

RN002 – Apenas administradores criam e gerenciam eventos

- Usuários comuns apenas votam.

RN003 – Pitchs só existem dentro de um evento

- Não pode haver pitch solto no sistema.

RN004 – Evento pode ser público ou privado

- Eventos privados acessados por chave.
- Eventos públicos acessados pelo link.

RN005 – Votos são irreversíveis

- Usuário não pode alterar o voto após confirmar.

RN006 – Dashboard deve atualizar automaticamente

- via JavaScript + fetch ou WebSocket (opcional)

🗃️ Modelo de Dados (Sugestão)

User (Django default, estendido)

- id
- name
- email
- password
- role (admin / participante)

Event

- id
- title
- description

- logo
- starts_at
- ends_at
- is_active
- access_code

Pitch

- id
- event_id
- title
- description
- team
- image

Vote

- id
- pitch_id
- user_id
- event_id
- created_at
- ip_address
- user_agent

⚙️ Skills Necessárias do Desenvolvedor

O desenvolvedor contratado deve dominar:

Backend

✔ Django (Views, ORM, Middlewares, Signals, Admin)
✔ Django Rest Framework (se API for usada)
✔ PostgreSQL + Modelagem de Banco
✔ Autenticação e autorização Django
✔ Segurança (CSRF, XSS, Injection, JWT)
✔ Django Admin personalizado

Frontend

✔ HTML + CSS (Tailwind/Bootstrap)
✔ JavaScript (fetch/AJAX)
✔ Template Engine Django

DevOps

✔ Docker (containers para produção)
✔ Deploy em Render/Heroku/AWS/Oracle
✔ Configuração HTTPS
✔ Logs e monitoramento

Extras (Desejável)

✔ WebSockets ou Server-Sent Events (para dashboard realtime)
✔ Redis (cache)