ESPECIFICAÇÕES TÉCNICAS - PORTAL DE ASSOCIADOS

INSTITUTO PORTUGUÊS DE NEGÓCIOS SOCIAIS – BUREAU SOCIAL

📋 VISÃO GERAL DO SISTEMA

Sistema web completo para gestão de associados, votações online e geração automática de atas de assembleia.

Objetivos Principais

1. **Autenticação segura** de associados
2. **Acesso a documentos** e atas de assembleia
3. **Sistema de votação online** simples e transparente
4. **Geração automática de atas** de assembleia
5. **Painel administrativo** para gestão

🏗️ ARQUITETURA DO SISTEMA

Stack Tecnológico Recomendado

Frontend:

- React 19 (já utilizado no site atual)
- TailwindCSS (design consistente)
- React Router (navegação)
- Axios (comunicação com API)

Backend:

- Flask (Python) - Simples e poderoso
- Flask-Login (autenticação)
- Flask-SQLAlchemy (banco de dados)
- Flask-CORS (integração frontend/backend)

Banco de Dados:

- PostgreSQL (produção) ou SQLite (desenvolvimento)

Geração de Documentos:

- python-docx (geração de atas em Word)
- ReportLab ou WeasyPrint (geração de PDFs)

Hospedagem:

- Frontend: Netlify/Vercel (gratuito)
- Backend: Railway/Render (gratuito com limitações)
- Banco de Dados: Supabase/Railway (gratuito)

📊 ESTRUTURA DO BANCO DE DADOS

Tabelas Principais

1. users (Associados)

SQL

```sql
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    nome VARCHAR(200) NOT NULL,
    email VARCHAR(200) UNIQUE NOT NULL,
    password_hash VARCHAR(255) NOT NULL,
    categoria VARCHAR(50) NOT NULL,  -- fundador, efetivo, contribuinte, etc.
    numero_socio VARCHAR(20) UNIQUE,
    telefone VARCHAR(20),
    data_adesao DATE,
    ativo BOOLEAN DEFAULT TRUE,
    is_admin BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

2. assemblies (Assembleias)

SQL

```sql
CREATE TABLE assemblies (
    id SERIAL PRIMARY KEY,
    titulo VARCHAR(200) NOT NULL,
```

```
    tipo VARCHAR(50) NOT NULL,  -- ordinaria, extraordinaria
    data_assembleia TIMESTAMP NOT NULL,
    local VARCHAR(200),
    convocatoria TEXT,
    ordem_dia TEXT,  -- JSON com itens da ordem do dia
    status VARCHAR(20) DEFAULT 'agendada',  -- agendada, em_curso, encerrada
    quorum_minimo INTEGER DEFAULT 50,  -- percentual
    ata_gerada BOOLEAN DEFAULT FALSE,
    ata_path VARCHAR(500),
    created_by INTEGER REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. voting_items (Itens de Votação)

SQL

```
CREATE TABLE voting_items (
    id SERIAL PRIMARY KEY,
    assembly_id INTEGER REFERENCES assemblies(id),
    ordem INTEGER NOT NULL,  -- ordem na assembleia
    titulo VARCHAR(200) NOT NULL,
    descricao TEXT,
    tipo VARCHAR(50) DEFAULT 'simples',  -- simples, qualificada, secreta
    opcoes JSON,  -- ['Aprovar', 'Rejeitar', 'Abstenção']
    quorum_necessario INTEGER DEFAULT 50,  -- percentual
    status VARCHAR(20) DEFAULT 'pendente',  -- pendente, aberta, encerrada
    data_abertura TIMESTAMP,
    data_encerramento TIMESTAMP,
    resultado JSON,  -- contagem de votos
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

4. votes (Votos)

SQL

```
CREATE TABLE votes (
    id SERIAL PRIMARY KEY,
    voting_item_id INTEGER REFERENCES voting_items(id),
    user_id INTEGER REFERENCES users(id),
    voto VARCHAR(50) NOT NULL,  -- Aprovar, Rejeitar, Abstenção
    justificativa TEXT,
    ip_address VARCHAR(50),
    voted_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
```

```sql
    UNIQUE(voting_item_id, user_id)  -- um voto por pessoa por item
);
```

5. documents (Documentos)

SQL

```sql
CREATE TABLE documents (
    id SERIAL PRIMARY KEY,
    titulo VARCHAR(200) NOT NULL,
    tipo VARCHAR(50) NOT NULL,  -- ata, regulamento, relatorio, etc.
    categoria VARCHAR(50),
    assembly_id INTEGER REFERENCES assemblies(id),
    file_path VARCHAR(500) NOT NULL,
    file_size INTEGER,
    visivel_para VARCHAR(20) DEFAULT 'todos',  -- todos, admin, direção
    uploaded_by INTEGER REFERENCES users(id),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

6. presences (Presenças)

SQL

```sql
CREATE TABLE presences (
    id SERIAL PRIMARY KEY,
    assembly_id INTEGER REFERENCES assemblies(id),
    user_id INTEGER REFERENCES users(id),
    presente BOOLEAN DEFAULT FALSE,
    representado_por INTEGER REFERENCES users(id),
    hora_entrada TIMESTAMP,
    hora_saida TIMESTAMP,
    UNIQUE(assembly_id, user_id)
);
```

7. notifications (Notificações)

SQL

```sql
CREATE TABLE notifications (
    id SERIAL PRIMARY KEY,
    user_id INTEGER REFERENCES users(id),
    tipo VARCHAR(50) NOT NULL,  -- assembleia, votacao, documento
    titulo VARCHAR(200) NOT NULL,
```

```
    mensagem TEXT,
    link VARCHAR(500),
    lida BOOLEAN DEFAULT FALSE,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

🔐 SISTEMA DE AUTENTICAÇÃO

Fluxo de Login

1. **Registro Inicial:**
   - Admin cria conta do associado com dados básicos
   - Sistema envia email com link de ativação
   - Associado define senha no primeiro acesso
2. **Login:**
   - Email + Senha
   - Geração de JWT token (validade: 24h)
   - Token armazenado no localStorage do navegador
3. **Níveis de Acesso:**
   - **Associado:** Acesso a documentos, votações, perfil
   - **Direção:** + Criar assembleias, ver resultados detalhados
   - **Admin:** + Gestão de usuários, configurações

Endpoints de Autenticação

Python

```
POST /api/auth/register          # Registro inicial (admin only)
POST /api/auth/activate          # Ativação de conta
POST /api/auth/login             # Login
POST /api/auth/logout            # Logout
POST /api/auth/forgot-password   # Recuperação de senha
POST /api/auth/reset-password    # Redefinir senha
GET  /api/auth/me                # Dados do usuário logado
```

📱 FUNCIONALIDADES DO PORTAL

1. DASHBOARD DO ASSOCIADO

Componentes:

- Resumo de votações pendentes
- Próximas assembleias
- Documentos recentes
- Notificações não lidas
- Status da quota (em dia/atraso)

Endpoints:

```python
GET /api/dashboard/summary       # Resumo geral
GET /api/dashboard/notifications # Notificações
```

2. GESTÃO DE ASSEMBLEIAS

Funcionalidades Admin/Direção:

- Criar nova assembleia
- Definir ordem do dia
- Adicionar itens de votação
- Convocar associados (email automático)
- Registrar presenças
- Encerrar assembleia
- Gerar ata automaticamente

Funcionalidades Associado:

- Ver assembleias agendadas
- Confirmar presença
- Indicar representante (procuração)
- Acessar convocatória
- Participar de votações

Endpoints:

```python
```

```
# Admin/Direção
POST   /api/assemblies                    # Criar assembleia
PUT    /api/assemblies/{id}               # Editar assembleia
DELETE /api/assemblies/{id}               # Cancelar assembleia
POST   /api/assemblies/{id}/convoke       # Enviar convocatória
POST   /api/assemblies/{id}/start         # Iniciar assembleia
POST   /api/assemblies/{id}/end           # Encerrar assembleia
POST   /api/assemblies/{id}/generate-ata  # Gerar ata

# Associados
GET    /api/assemblies                    # Listar assembleias
GET    /api/assemblies/{id}               # Detalhes da assembleia
POST   /api/assemblies/{id}/presence      # Confirmar presença
POST   /api/assemblies/{id}/proxy         # Indicar representante
```

3. SISTEMA DE VOTAÇÃO

Tipos de Votação:

- **Simples:** Maioria simples (50% + 1)
- **Qualificada:** 2/3 dos votos
- **Secreta:** Votos anônimos (apenas contagem)
- **Aberta:** Votos nominais (quem votou o quê)

Fluxo de Votação:

1. Admin cria item de votação
2. Sistema notifica associados
3. Associado acessa portal e vota
4. Voto é registrado (imutável)
5. Ao encerrar, sistema calcula resultado
6. Resultado é exibido em tempo real

Regras:

- Um voto por associado por item
- Voto não pode ser alterado após confirmação
- Apenas associados em dia com quotas podem votar
- Quorum mínimo deve ser atingido

Endpoints:

Python

```
# Admin/Direção
POST   /api/voting-items                  # Criar item de votação
PUT    /api/voting-items/{id}             # Editar item
POST   /api/voting-items/{id}/open        # Abrir votação
POST   /api/voting-items/{id}/close       # Encerrar votação
GET    /api/voting-items/{id}/results     # Resultados detalhados

# Associados
GET    /api/voting-items                  # Listar votações abertas
GET    /api/voting-items/{id}             # Detalhes da votação
POST   /api/voting-items/{id}/vote        # Votar
GET    /api/voting-items/{id}/my-vote     # Meu voto
```

4. GERAÇÃO AUTOMÁTICA DE ATAS

Estrutura da Ata:

Plain Text

```
ATA DA ASSEMBLEIA [TIPO] Nº [NÚMERO]
INSTITUTO PORTUGUÊS DE NEGÓCIOS SOCIAIS – BUREAU SOCIAL

Data: [DATA]
Hora: [HORA INÍCIO] - [HORA FIM]
Local: [LOCAL]

PRESENÇAS:
- Total de Associados: [NÚMERO]
- Presentes: [NÚMERO] ([PERCENTUAL]%)
- Representados: [NÚMERO]
- Quorum: [ATINGIDO/NÃO ATINGIDO]

Lista de Presentes:
[LISTA NOMINAL]

ORDEM DO DIA:
1. [ITEM 1]
2. [ITEM 2]
...

DELIBERAÇÕES:

1. [ITEM 1]
   Proposta: [DESCRIÇÃO]
   Votação:
   - A favor: [NÚMERO] ([PERCENTUAL]%)
```

```
   - Contra: [NÚMERO] ([PERCENTUAL]%)
   - Abstenções: [NÚMERO] ([PERCENTUAL]%)
   Resultado: [APROVADO/REJEITADO]

2. [ITEM 2]
   ...

ENCERRAMENTO:
Nada mais havendo a tratar, foi encerrada a assembleia às [HORA],
da qual se lavrou a presente ata que vai assinada pelos membros
da Mesa da Assembleia Geral.

_____________________________
Presidente da Mesa

_____________________________
Secretário da Mesa
```

Processo de Geração:

Python

```python
def gerar_ata(assembly_id):
    # 1. Buscar dados da assembleia
    assembly = Assembly.query.get(assembly_id)

    # 2. Buscar presenças
    presences = Presence.query.filter_by(assembly_id=assembly_id).all()

    # 3. Buscar votações e resultados
    voting_items = VotingItem.query.filter_by(assembly_id=assembly_id).all()

    # 4. Calcular estatísticas
    total_associados = User.query.filter_by(ativo=True).count()
    presentes = len([p for p in presences if p.presente])
    quorum = (presentes / total_associados) * 100

    # 5. Gerar documento Word
    doc = Document()

    # Cabeçalho
    doc.add_heading('ATA DA ASSEMBLEIA GERAL', 0)
    doc.add_paragraph(f'Data: {assembly.data_assembleia}')

    # Presenças
    doc.add_heading('PRESENÇAS', 1)
    doc.add_paragraph(f'Total: {presentes} ({quorum:.1f}%)')
```

```
    # Lista de presentes
    for presence in presences:
        if presence.presente:
            doc.add_paragraph(f'• {presence.user.nome}', style='List Bullet')

    # Ordem do dia
    doc.add_heading('ORDEM DO DIA', 1)
    for i, item in enumerate(voting_items, 1):
        doc.add_paragraph(f'{i}. {item.titulo}')

    # Deliberações
    doc.add_heading('DELIBERAÇÕES', 1)
    for i, item in enumerate(voting_items, 1):
        doc.add_heading(f'{i}. {item.titulo}', 2)
        doc.add_paragraph(f'Proposta: {item.descricao}')

        # Resultados da votação
        resultado = item.resultado  # JSON
        doc.add_paragraph('Votação:')
        doc.add_paragraph(f'  A favor: {resultado["aprovar"]}
({resultado["aprovar_pct"]:.1f}%)')
        doc.add_paragraph(f'  Contra: {resultado["rejeitar"]}
({resultado["rejeitar_pct"]:.1f}%)')
        doc.add_paragraph(f'  Abstenções: {resultado["abstencao"]}
({resultado["abstencao_pct"]:.1f}%)')
        doc.add_paragraph(f'Resultado: {resultado["status"].upper()}')

    # Encerramento
    doc.add_heading('ENCERRAMENTO', 1)
    doc.add_paragraph('Nada mais havendo a tratar...')

    # Salvar
    filename =
f'ata_assembleia_{assembly_id}_{datetime.now().strftime("%Y%m%d")}.docx'
    doc.save(filename)

    return filename
```

Endpoints:

Python

```
POST /api/assemblies/{id}/generate-ata    # Gerar ata
GET  /api/assemblies/{id}/ata             # Download da ata
```

5. GESTÃO DE DOCUMENTOS

Funcionalidades:

- Upload de documentos (atas, regulamentos, relatórios)
- Categorização automática
- Controle de acesso por categoria de associado
- Busca e filtros
- Download de documentos

Endpoints:

```python
GET    /api/documents                    # Listar documentos
GET    /api/documents/{id}               # Detalhes do documento
GET    /api/documents/{id}/download      # Download
POST   /api/documents                    # Upload (admin)
DELETE /api/documents/{id}               # Remover (admin)
```

6. PERFIL DO ASSOCIADO

Funcionalidades:

- Ver dados pessoais
- Atualizar email/telefone
- Alterar senha
- Ver histórico de votações
- Ver histórico de presenças
- Baixar comprovante de associado

Endpoints:

```python
GET  /api/profile                        # Dados do perfil
PUT  /api/profile                        # Atualizar perfil
PUT  /api/profile/password               # Alterar senha
GET  /api/profile/voting-history         # Histórico de votos
GET  /api/profile/presence-history       # Histórico de presenças
GET  /api/profile/certificate            # Comprovante PDF
```

🎨 INTERFACE DO USUÁRIO

Páginas Principais

1. Login / Ativação

- Formulário de login simples
- Link para recuperação de senha
- Página de ativação para novos associados

2. Dashboard

Plain Text

```
┌─────────────────────────────────────────────────┐
│ Bem-vindo, [Nome do Associado]                  │
├─────────────────────────────────────────────────┤
│                                                 │
│ 🗳️  Votações Pendentes (3)                      │
│ ├─ Aprovação do Orçamento 2026                  │
│ ├─ Eleição da Nova Direção                      │
│ └─ Alteração do Regulamento Interno             │
│                                                 │
│ 📅 Próximas Assembleias                         │
│ └─ Assembleia Geral Ordinária - 15/11/2025      │
│                                                 │
│ 📄 Documentos Recentes                          │
│ ├─ Ata AG Extraordinária - Set/2025             │
│ ├─ Relatório de Atividades Q3                   │
│ └─ Plano de Atividades 2026                     │
│                                                 │
│ 🔔 Notificações (2 não lidas)                   │
│                                                 │
└─────────────────────────────────────────────────┘
```

3. Assembleias

- Lista de assembleias (futuras, passadas)
- Filtros por tipo e data
- Detalhes da assembleia
- Botão de confirmação de presença
- Acesso à convocatória e ordem do dia

4. Votações

- Lista de votações abertas
- Detalhes da votação
- Interface de voto simples
- Confirmação de voto
- Resultados (após encerramento)

Exemplo de Interface de Votação:

Plain Text

```
┌─────────────────────────────────────────────────┐
│ Votação: Aprovação do Orçamento 2026            │
├─────────────────────────────────────────────────┤
│                                                 │
│ Descrição:                                      │
│ Proposta de orçamento para o ano de 2026...     │
│                                                 │
│ Seu Voto:                                       │
│ ○ Aprovar                                       │
│ ○ Rejeitar                                      │
│ ○ Abstenção                                     │
│                                                 │
│ Justificativa (opcional):                       │
│ [                                    ]          │
│                                                 │
│ [Confirmar Voto]                                │
│                                                 │
│ Status: 45 de 120 associados votaram (37.5%)    │
│ Encerramento: 10/11/2025 às 23:59               │
│                                                 │
└─────────────────────────────────────────────────┘
```

5. Documentos

- Grid de documentos com categorias
- Busca e filtros
- Preview de PDFs
- Download

6. Perfil

- Dados pessoais
- Histórico de participação
- Configurações de notificações

7. Admin - Gestão (apenas admin/direção)

- Criar assembleia
- Criar votação
- Gerenciar associados
- Upload de documentos
- Relatórios e estatísticas

🔔 SISTEMA DE NOTIFICAÇÕES

Tipos de Notificações

1. **Email:**
   - Convocatória de assembleia
   - Nova votação aberta
   - Lembrete de votação (24h antes do fim)
   - Resultado de votação
   - Novo documento disponível
2. **In-App:**
   - Badge de notificações não lidas
   - Lista de notificações no dashboard
   - Notificações em tempo real (WebSocket opcional)

Configuração de Emails

Python

```python
# Usando SendGrid ou SMTP
from flask_mail import Mail, Message

def enviar_convocatoria(assembly_id):
    assembly = Assembly.query.get(assembly_id)
    users = User.query.filter_by(ativo=True).all()
```

```
    for user in users:
        msg = Message(
            subject=f'Convocatória: {assembly.titulo}',
            recipients=[user.email],
            html=render_template('email/convocatoria.html',
                                 assembly=assembly,
                                 user=user)
        )
        mail.send(msg)
```

📊 RELATÓRIOS E ESTATÍSTICAS

Relatórios Disponíveis (Admin)

1. **Participação em Assembleias:**
   - Taxa de presença por assembleia
   - Associados mais participativos
   - Evolução da participação
2. **Votações:**
   - Taxa de participação por votação
   - Distribuição de votos
   - Tempo médio de votação
3. **Associados:**
   - Total de associados por categoria
   - Novos associados por mês
   - Taxa de retenção

Endpoints:

```
Python

GET /api/reports/participation          # Relatório de participação
GET /api/reports/voting-stats           # Estatísticas de votação
GET /api/reports/members                # Relatório de associados
```

🔒 SEGURANÇA

Medidas de Segurança

1. **Autenticação:**
   - Senhas com hash bcrypt
   - JWT tokens com expiração
   - Rate limiting em endpoints de login
2. **Autorização:**
   - Middleware de verificação de permissões
   - Roles: associado, direção, admin
   - Verificação de status ativo
3. **Votação:**
   - Verificação de unicidade de voto
   - Registro de IP e timestamp
   - Impossibilidade de alterar voto
   - Auditoria completa
4. **Dados:**
   - HTTPS obrigatório
   - Backup automático diário
   - Logs de auditoria
   - RGPD compliance

Exemplo de Middleware de Autorização

Python

```python
from functools import wraps
from flask import request, jsonify
import jwt

def token_required(f):
    @wraps(f)
    def decorated(*args, **kwargs):
        token = request.headers.get('Authorization')

        if not token:
            return jsonify({'message': 'Token não fornecido'}), 401

        try:
```

```
            data = jwt.decode(token, app.config['SECRET_KEY'], algorithms=
['HS256'])
            current_user = User.query.get(data['user_id'])
        except:
            return jsonify({'message': 'Token inválido'}), 401

        return f(current_user, *args, **kwargs)

    return decorated

def admin_required(f):
    @wraps(f)
    def decorated(current_user, *args, **kwargs):
        if not current_user.is_admin:
            return jsonify({'message': 'Acesso negado'}), 403

        return f(current_user, *args, **kwargs)

    return decorated
```

🚀 IMPLEMENTAÇÃO NO REPLIT

Estrutura de Pastas

Plain Text

```
bureau-social-portal/
├── backend/
│   ├── app.py                    # Aplicação Flask principal
│   ├── models.py                 # Modelos do banco de dados
│   ├── routes/
│   │   ├── auth.py              # Rotas de autenticação
│   │   ├── assemblies.py        # Rotas de assembleias
│   │   ├── voting.py            # Rotas de votação
│   │   ├── documents.py         # Rotas de documentos
│   │   └── profile.py           # Rotas de perfil
│   ├── utils/
│   │   ├── ata_generator.py    # Gerador de atas
│   │   ├── email_sender.py     # Envio de emails
│   │   └── validators.py       # Validações
│   ├── templates/               # Templates de email
│   └── requirements.txt         # Dependências Python
├── frontend/
│   ├── src/
│   │   ├── components/
```

```
│   │   │   ├── Auth/
│   │   │   ├── Dashboard/
│   │   │   ├── Assemblies/
│   │   │   ├── Voting/
│   │   │   ├── Documents/
│   │   │   └── Profile/
│   │   ├── App.jsx
│   │   └── main.jsx
│   ├── package.json
│   └── vite.config.js
├── .replit                      # Configuração do Replit
└── README.md
```

Arquivo .replit

Plain Text

```
run = "cd backend && python app.py"
language = "python3"

[nix]
channel = "stable-22_11"

[env]
FLASK_APP = "backend/app.py"
FLASK_ENV = "development"

[packager]
language = "python3"

[packager.features]
packageSearch = true
guessImports = true

[languages.python3]
pattern = "**/*.py"

[languages.python3.languageServer]
start = "pylsp"
```

requirements.txt

Plain Text

```
Flask==3.0.0
Flask-SQLAlchemy==3.1.1
```

```
Flask-Login==0.6.3
Flask-CORS==4.0.0
Flask-Mail==0.9.1
PyJWT==2.8.0
bcrypt==4.1.2
python-docx==1.1.0
reportlab==4.0.9
psycopg2-binary==2.9.9
python-dotenv==1.0.0
```

Comandos de Inicialização

Bash

```bash
# 1. Instalar dependências do backend
cd backend
pip install -r requirements.txt

# 2. Criar banco de dados
python
>>> from app import db
>>> db.create_all()
>>> exit()

# 3. Instalar dependências do frontend
cd ../frontend
npm install

# 4. Executar backend (terminal 1)
cd backend
python app.py

# 5. Executar frontend (terminal 2)
cd frontend
npm run dev
```

📝 CÓDIGO EXEMPLO - ESTRUTURA BÁSICA

backend/app.py

Python

```python
from flask import Flask, jsonify
from flask_sqlalchemy import SQLAlchemy
```

```python
from flask_cors import CORS
from flask_login import LoginManager
import os

app = Flask(__name__)
app.config['SECRET_KEY'] = os.environ.get('SECRET_KEY', 'dev-secret-key')
app.config['SQLALCHEMY_DATABASE_URI'] = os.environ.get('DATABASE_URL',
'sqlite:///bureau_social.db')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False

db = SQLAlchemy(app)
CORS(app)
login_manager = LoginManager(app)

# Importar modelos
from models import User, Assembly, VotingItem, Vote, Document

# Importar rotas
from routes.auth import auth_bp
from routes.assemblies import assemblies_bp
from routes.voting import voting_bp
from routes.documents import documents_bp
from routes.profile import profile_bp

# Registrar blueprints
app.register_blueprint(auth_bp, url_prefix='/api/auth')
app.register_blueprint(assemblies_bp, url_prefix='/api/assemblies')
app.register_blueprint(voting_bp, url_prefix='/api/voting-items')
app.register_blueprint(documents_bp, url_prefix='/api/documents')
app.register_blueprint(profile_bp, url_prefix='/api/profile')

@app.route('/')
def index():
    return jsonify({'message': 'Bureau Social Portal API', 'version':
'1.0.0'})

@app.route('/api/health')
def health():
    return jsonify({'status': 'healthy'})

if __name__ == '__main__':
    with app.app_context():
        db.create_all()
    app.run(host='0.0.0.0', port=5000, debug=True)
```

backend/routes/voting.py (Exemplo)

Python

```python
from flask import Blueprint, request, jsonify
from flask_login import login_required, current_user
from models import VotingItem, Vote, db
from datetime import datetime

voting_bp = Blueprint('voting', __name__)

@voting_bp.route('/', methods=['GET'])
@login_required
def list_voting_items():
    """Listar votações abertas"""
    items = VotingItem.query.filter_by(status='aberta').all()
    return jsonify([item.to_dict() for item in items])

@voting_bp.route('/<int:id>', methods=['GET'])
@login_required
def get_voting_item(id):
    """Detalhes de uma votação"""
    item = VotingItem.query.get_or_404(id)
    return jsonify(item.to_dict())

@voting_bp.route('/<int:id>/vote', methods=['POST'])
@login_required
def vote(id):
    """Registrar voto"""
    item = VotingItem.query.get_or_404(id)

    # Verificações
    if item.status != 'aberta':
        return jsonify({'error': 'Votação não está aberta'}), 400

    if not current_user.ativo:
        return jsonify({'error': 'Associado inativo'}), 403

    # Verificar se já votou
    existing_vote = Vote.query.filter_by(
        voting_item_id=id,
        user_id=current_user.id
    ).first()

    if existing_vote:
        return jsonify({'error': 'Você já votou neste item'}), 400

    # Registrar voto
    data = request.get_json()
    vote = Vote(
        voting_item_id=id,
```

```
        user_id=current_user.id,
        voto=data['voto'],
        justificativa=data.get('justificativa'),
        ip_address=request.remote_addr
    )

    db.session.add(vote)
    db.session.commit()

    return jsonify({'message': 'Voto registrado com sucesso'}), 201

@voting_bp.route('/<int:id>/results', methods=['GET'])
@login_required
def get_results(id):
    """Obter resultados da votação"""
    item = VotingItem.query.get_or_404(id)

    if item.status != 'encerrada':
        return jsonify({'error': 'Votação ainda não encerrada'}), 400

    # Calcular resultados
    votes = Vote.query.filter_by(voting_item_id=id).all()
    total = len(votes)

    results = {
        'total_votos': total,
        'aprovar': len([v for v in votes if v.voto == 'Aprovar']),
        'rejeitar': len([v for v in votes if v.voto == 'Rejeitar']),
        'abstencao': len([v for v in votes if v.voto == 'Abstenção'])
    }

    results['aprovar_pct'] = (results['aprovar'] / total * 100) if total > 0
else 0
    results['rejeitar_pct'] = (results['rejeitar'] / total * 100) if total >
0 else 0
    results['abstencao_pct'] = (results['abstencao'] / total * 100) if total
> 0 else 0

    # Determinar status
    if results['aprovar_pct'] > item.quorum_necessario:
        results['status'] = 'aprovado'
    else:
        results['status'] = 'rejeitado'

    return jsonify(results)
```

frontend/src/components/Voting/VotingCard.jsx (Exemplo)

JSX

```jsx
import React, { useState } from 'react';
import { Card, CardHeader, CardTitle, CardContent } from
'@/components/ui/card';
import { Button } from '@/components/ui/button';
import { RadioGroup, RadioGroupItem } from '@/components/ui/radio-group';
import { Label } from '@/components/ui/label';
import { Textarea } from '@/components/ui/textarea';
import axios from 'axios';

export function VotingCard({ votingItem, onVoteSuccess }) {
  const [selectedVote, setSelectedVote] = useState('');
  const [justificativa, setJustificativa] = useState('');
  const [isSubmitting, setIsSubmitting] = useState(false);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setIsSubmitting(true);

    try {
      await axios.post(
        `/api/voting-items/${votingItem.id}/vote`,
        {
          voto: selectedVote,
          justificativa: justificativa
        },
        {
          headers: {
            Authorization: `Bearer ${localStorage.getItem('token')}`
          }
        }
      );

      alert('Voto registrado com sucesso!');
      onVoteSuccess();
    } catch (error) {
      alert(error.response?.data?.error || 'Erro ao registrar voto');
    } finally {
      setIsSubmitting(false);
    }
  };

  return (
    <Card>
      <CardHeader>
        <CardTitle>{votingItem.titulo}</CardTitle>
```

```
      </CardHeader>
      <CardContent>
        <p className="mb-4 text-muted-foreground">{votingItem.descricao}</p>

        <form onSubmit={handleSubmit} className="space-y-4">
          <div>
            <Label className="mb-2 block">Seu Voto:</Label>
            <RadioGroup value={selectedVote} onValueChange={setSelectedVote}>
              <div className="flex items-center space-x-2">
                <RadioGroupItem value="Aprovar" id="aprovar" />
                <Label htmlFor="aprovar">Aprovar</Label>
              </div>
              <div className="flex items-center space-x-2">
                <RadioGroupItem value="Rejeitar" id="rejeitar" />
                <Label htmlFor="rejeitar">Rejeitar</Label>
              </div>
              <div className="flex items-center space-x-2">
                <RadioGroupItem value="Abstenção" id="abstencao" />
                <Label htmlFor="abstencao">Abstenção</Label>
              </div>
            </RadioGroup>
          </div>

          <div>
            <Label htmlFor="justificativa">Justificativa (opcional):</Label>
            <Textarea
              id="justificativa"
              value={justificativa}
              onChange={(e) => setJustificativa(e.target.value)}
              placeholder="Explique seu voto (opcional)..."
              rows={3}
            />
          </div>

          <Button
            type="submit"
            disabled={!selectedVote || isSubmitting}
            className="w-full"
          >
            {isSubmitting ? 'Enviando...' : 'Confirmar Voto'}
          </Button>
        </form>

        <div className="mt-4 text-sm text-muted-foreground">
          <p>Status: {votingItem.votos_registrados} de
{votingItem.total_associados} votaram</p>
          <p>Encerramento: {new
Date(votingItem.data_encerramento).toLocaleString('pt-PT')}</p>
```

```
          </div>
        </CardContent>
      </Card>
    );
}
```

📅 CRONOGRAMA DE IMPLEMENTAÇÃO

Fase 1: Fundação (2 semanas)

- ✅ Configurar ambiente Replit
- ✅ Criar estrutura de banco de dados
- ✅ Implementar sistema de autenticação
- ✅ Criar interface de login/registro

Fase 2: Gestão de Assembleias (2 semanas)

- ✅ CRUD de assembleias
- ✅ Sistema de convocatória
- ✅ Registro de presenças
- ✅ Interface de gestão de assembleias

Fase 3: Sistema de Votação (2 semanas)

- ✅ CRUD de itens de votação
- ✅ Lógica de votação
- ✅ Cálculo de resultados
- ✅ Interface de votação

Fase 4: Geração de Atas (1 semana)

- ✅ Lógica de geração automática
- ✅ Template de ata
- ✅ Geração em Word e PDF
- ✅ Armazenamento e download

Fase 5: Documentos e Perfil (1 semana)

- ✅ Sistema de upload de documentos
- ✅ Gestão de perfil
- ✅ Histórico de participação

Fase 6: Notificações e Emails (1 semana)

- ✅ Sistema de notificações in-app
- ✅ Envio de emails automáticos
- ✅ Templates de email

Fase 7: Admin e Relatórios (1 semana)

- ✅ Painel administrativo
- ✅ Relatórios e estatísticas
- ✅ Gestão de associados

Fase 8: Testes e Deploy (1 semana)

- ✅ Testes de segurança
- ✅ Testes de usabilidade
- ✅ Deploy em produção
- ✅ Documentação final

Total: 11 semanas

💰 ESTIMATIVA DE CUSTOS

Opção Gratuita (Limitada)

- **Replit:** Gratuito (com limitações de CPU)
- **Supabase:** Gratuito até 500MB
- **Netlify/Vercel:** Gratuito
- **SendGrid:** 100 emails/dia grátis
- **Total:** €0/mês

Opção Profissional (Recomendada)

- **Railway:** €5/mês (backend)

- **Supabase Pro:** €25/mês (banco de dados)
- **Netlify Pro:** €19/mês (frontend)
- **SendGrid Essentials:** €15/mês (40.000 emails)
- **Total:** €64/mês (~€768/ano)

🎯 PRÓXIMOS PASSOS

1. **Revisar especificações** com a Direção
2. **Aprovar funcionalidades** e prioridades
3. **Configurar ambiente** no Replit
4. **Iniciar desenvolvimento** seguindo o cronograma
5. **Testes com grupo piloto** de associados
6. **Lançamento oficial** do portal

📞 SUPORTE TÉCNICO

Para dúvidas ou suporte durante a implementação:

- **Email:** diego@greencheck.pt
- **Telefone:** +351 931 721 901

Documento elaborado para o Instituto Português de Negócios Sociais – Bureau Social
Data: Outubro 2025 **Versão:** 1.0