Sistema de Gestão de Materiais - Backend em Express

Vou criar um servidor Express organizado no padrão MVC para atender aos requisitos do sistema. O foco será no MVP (Minimum Viable Product) com as funcionalidades essenciais.

Estrutura do Projeto

```
/src
  /config
    db.js          # Configuração do banco de dados
    auth.js        # Configuração de autenticação
  /controllers
    authController.js
    setorController.js
    usuarioController.js
    materialController.js
    solicitacaoController.js
    planejamentoController.js
  /middlewares
    authMiddleware.js
    errorMiddleware.js
  /models
    Setor.js
    Usuario.js
    Material.js
    Solicitacao.js
    ItemSolicitacao.js
    Planejamento.js
    ItemPlanejamento.js
  /routes
    index.js
    authRoutes.js
    setorRoutes.js
    usuarioRoutes.js
    materialRoutes.js
    solicitacaoRoutes.js
    planejamentoRoutes.js
  /utils
    apiResponse.js # Padronização de respostas
  app.js           # Aplicação principal
  server.js        # Inicialização do servidor
```

1. Configuração Inicial

app.js

```
const express = require('express');
const cors = require('cors');
const morgan = require('morgan');
const helmet = require('helmet');
const cookieParser = require('cookie-parser');

// Importar rotas
const routes = require('./routes');

// Criar aplicação Express
const app = express();

// Middlewares básicos
app.use(helmet());
app.use(cors({
  origin: process.env.FRONTEND_URL || 'http://localhost:3000',
  credentials: true
}));
app.use(morgan('dev'));
app.use(express.json());
app.use(cookieParser());

// Conectar ao banco de dados
require('./config/db');

// Configurar rotas
app.use('/api', routes);

// Middleware de erro (deve ser o último)
const errorMiddleware = require('./middlewares/errorMiddleware');
app.use(errorMiddleware);

module.exports = app;
```

server.js

```
const app = require('./app');
const PORT = process.env.PORT || 5000;

app.listen(PORT, () => {
  console.log(`Servidor rodando na porta ${PORT}`);
});
```

config/db.js

```
const { Pool } = require('pg');
require('dotenv').config();

const pool = new Pool({
  user: process.env.DB_USER || 'postgres',
  host: process.env.DB_HOST || 'localhost',
  database: process.env.DB_NAME || 'gestao_materiais',
  password: process.env.DB_PASSWORD || 'postgres',
  port: process.env.DB_PORT || 5432,
});

// Testar conexão
pool.query('SELECT NOW()', (err) => {
  if (err) {
    console.error('Erro ao conectar ao PostgreSQL:', err);
  } else {
    console.log('Conectado ao PostgreSQL com sucesso');
  }
});

module.exports = {
  query: (text, params) => pool.query(text, params),
};
```

2. Autenticação e Autorização

config/auth.js

```
const jwt = require('jsonwebtoken');
require('dotenv').config();

const JWT_SECRET = process.env.JWT_SECRET || 'segredo_super_secreto';
const JWT_EXPIRES_IN = process.env.JWT_EXPIRES_IN || '1d';

// Gerar token JWT
const generateToken = (matricula, tipoUsuario) => {
  return jwt.sign({ matricula, tipoUsuario }, JWT_SECRET, {
    expiresIn: JWT_EXPIRES_IN,
  });
};

// Verificar token JWT
const verifyToken = (token) => {
  try {
    return jwt.verify(token, JWT_SECRET);
```

```
  } catch (err) {
    return null;
  }
};

module.exports = {
  generateToken,
  verifyToken,
  JWT_SECRET,
};
```

middlewares/authMiddleware.js

```
const { verifyToken } = require('../config/auth');
const ApiResponse = require('../utils/apiResponse');

// Middleware para verificar autenticação
const authRequired = (req, res, next) => {
  const token = req.cookies.token || req.headers['authorization']?.split(' ')[1];

  if (!token) {
    return res.status(401).json(ApiResponse.error('Token de autenticação não fornecido'));
  }

  const decoded = verifyToken(token);
  if (!decoded) {
    return res.status(401).json(ApiResponse.error('Token inválido ou expirado'));
  }

  req.user = decoded;
  next();
};

// Middleware para verificar tipo de usuário
const checkUserType = (...allowedTypes) => {
  return (req, res, next) => {
    if (!allowedTypes.includes(req.user.tipoUsuario)) {
      return res.status(403).json(ApiResponse.error('Acesso não autorizado'));
    }
    next();
  };
};

module.exports = {
  authRequired,
  checkUserType,
```

```
};
```

3. Modelos (Models)

models/Usuario.js

```
const db = require('../config/db');
const bcrypt = require('bcryptjs');
const ApiResponse = require('../utils/apiResponse');

class Usuario {
  // Criar usuário
  static async criar(matricula, nome, login, senha, email, tipoUsuario, idSetor) {
    try {
      const senhaHash = await bcrypt.hash(senha, 10);
      const result = await db.query(
        'INSERT INTO usuario (matricula, nome_usuario, login, senha_hash, email, tipo_usuari
        [matricula, nome, login, senhaHash, email, tipoUsuario, idSetor]
      );
      return ApiResponse.success(result.rows[0]);
    } catch (err) {
      return ApiResponse.error(err.message);
    }
  }

  // Buscar por matrícula
  static async buscarPorMatricula(matricula) {
    try {
      const result = await db.query('SELECT * FROM usuario WHERE matricula = $1', [matricula
      if (result.rows.length === 0) {
        return ApiResponse.error('Usuário não encontrado', 404);
      }
      return ApiResponse.success(result.rows[0]);
    } catch (err) {
      return ApiResponse.error(err.message);
    }
  }

  // Verificar credenciais
  static async verificarCredenciais(login, senha) {
    try {
      const result = await db.query('SELECT * FROM usuario WHERE login = $1', [login]);
      if (result.rows.length === 0) {
        return ApiResponse.error('Credenciais inválidas', 401);
      }
```

```
      const usuario = result.rows[0];
      const senhaValida = await bcrypt.compare(senha, usuario.senha_hash);
      if (!senhaValida) {
        return ApiResponse.error('Credenciais inválidas', 401);
      }

      return ApiResponse.success(usuario);
    } catch (err) {
      return ApiResponse.error(err.message);
    }
  }

  // Listar por setor
  static async listarPorSetor(idSetor) {
    try {
      const result = await db.query('SELECT * FROM usuario WHERE id_setor = $1', [idSetor]);
      return ApiResponse.success(result.rows);
    } catch (err) {
      return ApiResponse.error(err.message);
    }
  }
}

module.exports = Usuario;
```

4. Controladores (Controllers)

controllers/authController.js

```
const Usuario = require('../models/Usuario');
const { generateToken } = require('../config/auth');
const ApiResponse = require('../utils/apiResponse');

class AuthController {
  // Login
  static async login(req, res) {
    const { login, senha } = req.body;

    const result = await Usuario.verificarCredenciais(login, senha);
    if (!result.success) {
      return res.status(result.statusCode || 401).json(result);
    }

    const usuario = result.data;
    const token = generateToken(usuario.matricula, usuario.tipo_usuario);
```

```
    res.cookie('token', token, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'strict',
      maxAge: 24 * 60 * 60 * 1000, // 1 dia
    });

    return res.json(ApiResponse.success({
      matricula: usuario.matricula,
      nome: usuario.nome_usuario,
      tipoUsuario: usuario.tipo_usuario,
      token,
    }));
  }

  // Logout
  static logout(req, res) {
    res.clearCookie('token');
    return res.json(ApiResponse.success('Logout realizado com sucesso'));
  }

  // Obter informações do usuário logado
  static async getMe(req, res) {
    const result = await Usuario.buscarPorMatricula(req.user.matricula);
    if (!result.success) {
      return res.status(result.statusCode || 404).json(result);
    }

    const usuario = result.data;
    return res.json(ApiResponse.success({
      matricula: usuario.matricula,
      nome: usuario.nome_usuario,
      email: usuario.email,
      tipoUsuario: usuario.tipo_usuario,
      idSetor: usuario.id_setor,
    }));
  }
}

module.exports = AuthController;
```

controllers/solicitacaoController.js

```
const db = require('../config/db');
const ApiResponse = require('../utils/apiResponse');
```

```
class SolicitacaoController {
  // Criar solicitação
  static async criar(req, res) {
    const { itens } = req.body;
    const { matricula } = req.user;
    const idSetor = req.user.idSetor; // Assumindo que o middleware adiciona isso

    try {
      await db.query('BEGIN');

      // Inserir solicitação
      const solicitacaoResult = await db.query(
        `INSERT INTO solicitacao
         (data_solicitacao, status_solicitacao, id_setor, matricula_solicitante)
         VALUES (NOW(), 'Pendente', $1, $2)
         RETURNING id_solicitacao`,
        [idSetor, matricula]
      );

      const idSolicitacao = solicitacaoResult.rows[0].id_solicitacao;

      // Inserir itens da solicitação
      for (const item of itens) {
        await db.query(
          `INSERT INTO item_solicitacao
           (id_solicitacao, id_material, quantidade_solicitada)
           VALUES ($1, $2, $3)`,
          [idSolicitacao, item.idMaterial, item.quantidade]
        );
      }

      await db.query('COMMIT');
      return res.json(ApiResponse.success({ idSolicitacao }));
    } catch (err) {
      await db.query('ROLLBACK');
      return res.status(500).json(ApiResponse.error(err.message));
    }
  }

  // Listar solicitações do setor
  static async listarPorSetor(req, res) {
    const { idSetor } = req.user;

    try {
      const result = await db.query(
        `SELECT s.*, u.nome_usuario as solicitante_nome
```

```
          FROM solicitacao s
          JOIN usuario u ON s.matricula_solicitante = u.matricula
          WHERE s.id_setor = $1
          ORDER BY s.data_solicitacao DESC`,
          [idSetor]
        );
        return res.json(ApiResponse.success(result.rows));
      } catch (err) {
        return res.status(500).json(ApiResponse.error(err.message));
      }
    }

    // Aprovar/reprovar solicitação (apenas para Controle de Materiais)
    static async avaliar(req, res) {
      const { id } = req.params;
      const { status, motivo } = req.body;
      const { matricula } = req.user;

      if (!['Aprovada', 'Reprovada'].includes(status)) {
        return res.status(400).json(ApiResponse.error('Status inválido'));
      }

      try {
        const result = await db.query(
          `UPDATE solicitacao
           SET status_solicitacao = $1,
               motivo_reprovacao = $2,
               data_aprovacao_reprovacao = NOW(),
               matricula_aprovador = $3
           WHERE id_solicitacao = $4
           RETURNING *`,
          [status, motivo, matricula, id]
        );

        if (result.rowCount === 0) {
          return res.status(404).json(ApiResponse.error('Solicitação não encontrada'));
        }

        return res.json(ApiResponse.success(result.rows[0]));
      } catch (err) {
        return res.status(500).json(ApiResponse.error(err.message));
      }
    }
}

module.exports = SolicitacaoController;
```

5. Rotas

routes/authRoutes.js

```
const express = require('express');
const router = express.Router();
const AuthController = require('../controllers/authController');
const { authRequired } = require('../middlewares/authMiddleware');

router.post('/login', AuthController.login);
router.post('/logout', authRequired, AuthController.logout);
router.get('/me', authRequired, AuthController.getMe);

module.exports = router;
```

routes/solicitacaoRoutes.js

```
const express = require('express');
const router = express.Router();
const SolicitacaoController = require('../controllers/solicitacaoController');
const { authRequired, checkUserType } = require('../middlewares/authMiddleware');

// Rotas para coordenadores
router.post('/', authRequired, checkUserType('Coordenador'), SolicitacaoController.criar);
router.get('/setor', authRequired, checkUserType('Coordenador'), SolicitacaoController.lista

// Rotas para controle de materiais
router.put('/:id/avaliar', authRequired, checkUserType('ControleMateriais'), SolicitacaoCont

module.exports = router;
```

routes/index.js

```
const express = require('express');
const router = express.Router();

// Importar todas as rotas
const authRoutes = require('./authRoutes');
const setorRoutes = require('./setorRoutes');
const usuarioRoutes = require('./usuarioRoutes');
const materialRoutes = require('./materialRoutes');
const solicitacaoRoutes = require('./solicitacaoRoutes');
const planejamentoRoutes = require('./planejamentoRoutes');

// Configurar rotas
router.use('/auth', authRoutes);
router.use('/setores', setorRoutes);
```

```
router.use('/usuarios', usuarioRoutes);
router.use('/materiais', materialRoutes);
router.use('/solicitacoes', solicitacaoRoutes);
router.use('/planejamentos', planejamentoRoutes);

module.exports = router;
```

6. Utilitários

utils/apiResponse.js

```
class ApiResponse {
  static success(data, message = 'Operação realizada com sucesso') {
    return {
      success: true,
      message,
      data,
    };
  }

  static error(message = 'Ocorreu um erro', statusCode = 400) {
    return {
      success: false,
      message,
      statusCode,
    };
  }
}

module.exports = ApiResponse;
```

Funcionalidades Implementadas (MVP)

1. **Autenticação e Autorização**:
   - Login com JWT
   - Proteção de rotas por tipo de usuário
   - Gestão de sessão com cookies seguros
2. **Gestão de Solicitações**:
   - Criação de solicitações com múltiplos itens (Coordenador)
   - Listagem de solicitações por setor
   - Aprovação/reprovação (Controle de Materiais)
3. **Fluxo Básico**:
   - Coordenador cria solicitação → Pendente
   - Controle de Materiais avalia → Aprovada/Reprovada
   - Histórico de solicitações
4. **Estrutura MVC**:

- Models para interação com o banco
- Controllers para lógica de negócio
- Rotas organizadas por funcionalidade

5. **Pronto para Frontend**:
   - Todas as respostas em JSON
   - Padronização de respostas (success/error)
   - CORS configurado

Como Expandir

1. Adicionar endpoints para:
   - Catálogo de materiais (CRUD completo)
   - Planejamento de demanda
   - Histórico e relatórios
   - Notificações
2. Implementar:
   - Validações mais robustas
   - Paginação para listagens
   - Filtros avançados
   - Exportação de dados (CSV/XML)
3. Melhorar:
   - Tratamento de erros
   - Logging detalhado
   - Documentação da API (Swagger/OpenAPI)

Este MVP cobre os requisitos essenciais do sistema e estabelece uma base sólida para expansão futura.