Guia de Implementação Multi-Empresa (Multi-Tenant)

📋 Visão Geral

Implementar isolamento de dados por empresa em todas as operações do sistema.

🔧 ETAPA 1: Middleware de Contexto

1.1 Criar `src/middlewares/empresaContext.js`

```javascript
const ApiError = require('../utils/apiError');

const setEmpresaContext = (req, res, next) => {
  if (!req.usuario) {
    throw new ApiError(401, 'Usuário não autenticado');
  }

  req.empresaId = req.usuario.empresaId;
  next();
};

module.exports = { setEmpresaContext };
```

🔧 ETAPA 2: Atualizar Repositories

2.1 alunoRepository.js

- **Método** `criar`: Adicionar `empresaId: data.empresaId` no `prisma.aluno.create`

- **Método** `buscarTodos`: Adicionar `if (empresaId) where.empresaId = empresaId` antes do Promise.all
- **Métodos com** `findFirst/findUnique`: Adicionar `empresaId` no `where`

2.2 caixaRepository.js

- Adicionar `empresaId` em todos os métodos `create`, `findMany`, `findFirst`, `findUnique`

2.3 contaPagarRepository.js e contaReceberRepository.js

- Adicionar filtro `where.empresaId = empresaId` em `buscarTodos`
- Adicionar `empresaId` nos métodos `findUnique`, `groupBy`

2.4 descontoRepository.js, funcaoRepository.js, localRepository.js, planoRepository.js, turmaRepository.js

- Mesmo padrão: adicionar `empresaId` em `where` de todos os métodos

2.5 frequenciaRepository.js

- Adicionar `empresaId` no filtro `where` de `buscarTodos` e `buscarPorAlunoEData`

2.6 funcionarioRepository.js

- Adicionar `empresaId` em todos os `where`

2.7 matriculaRepository.js

- Adicionar `empresaId` nos filtros

2.8 pessoaRepository.js

- Adicionar `empresaId` em `create` e todos os `where`
- **Importante**: Alterar `@@unique([empresaId, codigo])` no schema

🔧 ETAPA 3: Atualizar Services

3.1 alunoService.js

- **Método** `criarComPessoa`:
  - Adicionar `empresaId: aluno.empresaId` ao criar pessoa
  - Adicionar `empresaId: aluno.empresaId` ao criar aluno
- **Método** `listarTodos`: Adicionar `empresaId` nos filtros

3.2 caixaService.js

- **Método** `criar`: Adicionar `empresaId: data.empresaId`
- **Métodos** `buscarAberto`, `listarTodos`: Adicionar empresaId no filtro

3.3 contaPagarService.js e contaReceberService.js

- **Método** `criar`: Adicionar `empresaId: data.empresaId`
- **Métodos de busca**: Adicionar empresaId nos filtros

3.4 descontoService.js, funcaoService.js, localService.js, planoService.js, turmaService.js

- **Método** `criar`: Adicionar `empresaId: data.empresaId`
- **Métodos de busca**: Adicionar empresaId nos filtros

3.5 frequenciaService.js

- **Método** `registrar`: Adicionar `empresaId: dados.empresaId`

3.6 funcionarioService.js

- **Método** `criarComPessoa`: Adicionar empresaId ao criar pessoa e funcionário

3.7 matriculaService.js

- **Método** `criar`: Adicionar `empresaId: data.empresaId`

3.8 pessoaService.js

- **Método** `gerarProximoCodigo`: Adicionar filtro `where: { empresaId }`
- **Método** `criar`: Adicionar `empresaId: data.empresaId`

🔧 ETAPA 4: Atualizar Controllers

4.1 Todos os Controllers

Adicionar `empresaId: req.empresaId` nos métodos que criam/atualizam:

```javascript
// Exemplo em alunoController.js
criarComPessoa = asyncHandler(async (req, res) => {
  const dadosCompletos = {
    ...req.body,
    aluno: {
      ...req.body.aluno,
      empresaId: req.empresaId
    }
  };

  const aluno = await alunoService.criarComPessoa(dadosCompletos);
  // ...
});
```

Aplicar em:

- alunoController.js
- caixaController.js
- contaPagarController.js
- contaReceberController.js
- descontoController.js
- frequenciaController.js
- funcaoController.js
- funcionarioController.js
- localController.js
- matriculaController.js
- pessoaController.js
- planoController.js
- turmaController.js
- visitanteController.js

🔧 ETAPA 5: Atualizar Rotas

5.1 Todas as rotas em `src/routes/`

Adicionar middleware após autenticação:

```
javascript
```

```
const { verificarAutenticacao } = require('../middlewares/auth');
const { setEmpresaContext } = require('../middlewares/empresaContext');

// Aplicar em TODAS as rotas protegidas
router.use(verificarAutenticacao);
router.use(setEmpresaContext);

// Rotas...
router.get('/', controller.listarTodos);
```

Aplicar em:

- alunoRoutes.js
- caixaRoutes.js
- contaPagarRoutes.js
- contaReceberRoutes.js
- descontoRoutes.js
- frequenciaRoutes.js
- funcaoRoutes.js
- funcionarioRoutes.js
- localRoutes.js
- matriculaRoutes.js
- pessoaRoutes.js
- planoRoutes.js

- turmaRoutes.js
- visitanteRoutes.js

🔧 ETAPA 6: Atualizar Jobs

6.1 gerarCobrancasRecorrentes.js

javascript

```javascript
// Linha ~25 (buscarMatriculasParaGerar)
where: {
  empresaId: req.empresaId, // ← Adicionar
  situacao: 'ATIVA',
  // ...
}
```

6.2 frequenciaJob.js

javascript

```javascript
// Linha ~15 (buscarTodos)
const { alunos } = await alunoRepository.buscarTodos({
  empresaId: req.empresaId, // ← Adicionar
  take: 1000
});
```

🔧 ETAPA 7: Atualizar Schema Prisma

7.1 schema_multiempresa.txt

Adicionar `empresaId` em TODOS os models (já está no schema fornecido)

7.2 Executar migração

```bash
npx prisma generate
npx prisma db push
```

✅ CHECKLIST DE VALIDAÇÃO

- [ ] Todas as queries incluem `empresaId` no filtro
- [ ] Todas as criações incluem `empresaId`
- [ ] Middleware `setEmpresaContext` está em todas as rotas protegidas
- [ ] Token JWT inclui `empresaId`
- [ ] Schema Prisma atualizado com indexes `@@index([empresaId])`
- [ ] Testes realizados com múltiplas empresas

🚨 PONTOS CRÍTICOS

1. **Nunca expor dados de outra empresa**: Sempre validar `empresaId`
2. **Códigos únicos por empresa**: Usar `@@unique([empresaId, codigo])`
3. **Jobs agendados**: Processar empresa por empresa
4. **Relatórios**: Filtrar sempre por empresaId

📊 EXEMPLO DE TESTE

```javascript
// Criar 2 empresas
// Criar 1 usuário em cada
// Logar com usuário 1
// Criar aluno
// Logar com usuário 2
// Tentar buscar aluno do usuário 1 → Deve retornar vazio
```

Implementação completa garante isolamento total entre empresas no sistema multi-tenant.