AC0 - Estrutura Inicial do Projeto

Projeto: Software Product

Instituição: Faculdade Impacta

Data: 15/02/2026

Versão: 1.0

🎯 OBJETIVO DO AC0

Estabelecer a **estrutura base** do projeto antes do desenvolvimento das funcionalidades, garantindo organização, versionamento e ambiente configurado.

📦 ENTREGAS DO AC0

🗄️ BANCO DE DADOS

- **Plataforma:** SQL Server 2019+
- **Banco:** `SoftwareProduct`
- **Tabelas:** `dbo.Usuarios`
- **Recursos:** PKs, Unique Constraints, Triggers, Índices

⚙️ BACKEND

- **Stack:** Node.js + Express.js + SQL Server
- **Arquitetura:** 3 camadas (Routes → Controllers → Repositories)
- **Módulo:** Usuários (CRUD completo + Login)

🎨 FRONTEND

- **Stack:** HTML5 + CSS3 + JavaScript Vanilla
- **Páginas:** Login + Dashboard (estático)
- **Features:** Autenticação, proteção de rotas, design responsivo

---

📁 ESTRUTURA FINAL DO PROJETO

```
C:\\software-product\\
├── sql/                              ← Scripts SQL
│   ├── CreateSoftwareProduct.sql
│   ├── Usuarios.sql
│   └── TesteUsuarios.sql
├── src/                              ← Backend
│   ├── config/
│   │   └── database.js
│   ├── repositories/
│   │   └── usuarioRepository.js
│   ├── controllers/
│   │   └── usuarioController.js
│   └── routes/
│       └── usuarioRoutes.js
├── public/                           ← Frontend
│   ├── pages/
│   │   ├── login.html
│   │   └── dashboard.html
│   └── assets/
│       ├── css/
│       │   ├── style.css
│       │   ├── login.css
│       │   └── dashboard.css
│       └── js/
│           ├── config.js
│           ├── auth.js
│           └── login.js
├── .env                              ← Configurações sensíveis
├── .gitignore                        ← Arquivos ignorados pelo Git
├── server.js                         ← Servidor Node.js
├── package.json                      ← Dependências do projeto
└── README.md                         ← Documentação
```

🔢 ORDEM DE INSTALAÇÃO (PASSO A PASSO)

FASE 1: BANCO DE DADOS

1.1 - Criar Banco e Tabelas no SQL Server

```
-- Executar na ordem:
1. CreateSoftwareProduct.sql  → Cria o banco
2. Usuarios.sql               → Cria a tabela
3. TesteUsuarios.sql          → Testa CRUD
```

✅ **Checkpoint:** Tabela `dbo.Usuarios` criada e testada.

FASE 2: ESTRUTURA DE PASTAS

2.1 - Criar Estrutura de Pastas

```
# Criar pastas manualmente ou via terminal:
mkdir sql src public
mkdir src\\config src\\repositories src\\controllers src\\r
outes
mkdir public\\pages public\\assets
mkdir public\\assets\\css public\\assets\\js
```

✅ **Checkpoint:** Pastas criadas conforme árvore do projeto.

FASE 3: ARQUIVOS DE CONFIGURAÇÃO

3.1 - Criar .gitignore

Por quê? Evitar enviar arquivos sensíveis e desnecessários ao GitHub.

Arquivo: `.gitignore`

```
# Dependências
node_modules/

# Configurações sensíveis
.env
```

```
# Logs
*.log
npm-debug.log*

# Sistema operacional
.DS_Store
Thumbs.db

# IDEs
.vscode/
.idea/
```

✅ **Checkpoint:** `.gitignore` criado na raiz do projeto.

3.2 - Inicializar package.json

Por quê? Gerenciar dependências do Node.js.

```
npm init -y
```

Resultado: Arquivo `package.json` criado automaticamente.

Editar manualmente para:

```
{
  "name": "software-product",
  "version": "1.0.0",
  "description": "Sistema de gerenciamento acadêmico - AC
1",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js"
  },
  "keywords": ["gestao", "impacta", "sql-server"],
  "author": "Buselli Rogerio",
  "license": "ISC",
  "dependencies": {
    "express": "^4.18.2",
```

```
    "mssql": "^10.0.1",
    "dotenv": "^16.3.1",
    "cors": "^2.8.5",
    "bcrypt": "^5.1.1"
  },
  "devDependencies": {
    "nodemon": "^3.0.1"
  }
}
```

✅ **Checkpoint:** `package.json` configurado.

3.3 - Instalar Dependências

Por quê? Baixar bibliotecas necessárias para o backend.

```
# Dependências principais
npm install express mssql dotenv cors bcrypt

# Dependência de desenvolvimento (opcional)
npm install --save-dev nodemon
```

O que cada uma faz:

- `express` → Framework web
- `mssql` → Conexão com SQL Server
- `dotenv` → Variáveis de ambiente
- `cors` → Permitir requisições do frontend
- `bcrypt` → Criptografia de senhas
- `nodemon` → Reinicia servidor automaticamente (DEV)

✅ **Checkpoint:** Pasta `node_modules/` criada automaticamente.

3.4 - Criar .env

Por quê? Armazenar credenciais do banco de forma segura.

Arquivo: `.env`

```
# Configurações do Banco de Dados
DB_SERVER=localhost
DB_DATABASE=SoftwareProduct
DB_USER=seu_usuario_sql
DB_PASSWORD=sua_senha_sql

# Porta do Servidor
PORT=3000
```

⚠️ **IMPORTANTE:** Nunca versionar este arquivo (já está no `.gitignore`).

✅ **Checkpoint:** `.env` criado e configurado.

FASE 4: BACKEND (Arquivos JavaScript)

4.1 - Criar database.js

Local: `src/config/database.js`

```
const sql = require('mssql');
require('dotenv').config();

const config = {
    server: process.env.DB_SERVER,
    database: process.env.DB_DATABASE,
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    options: {
        encrypt: false,
        trustServerCertificate: true
    }
};

module.exports = config;
```

4.2 - Criar usuarioRepository.js

Local: `src/repositories/usuarioRepository.js`

```
const sql = require('mssql');
const config = require('../config/database');

class UsuarioRepository {
    async login(login, senha) {
        const pool = await sql.connect(config);
        const result = await pool.request()
            .input('login', sql.NVarChar, login)
            .input('senha', sql.NVarChar, senha)
            .query('SELECT * FROM dbo.Usuarios WHERE Login
= @login AND Senha = @senha AND Ativo = 1');
        return result.recordset[0];
    }

    async listarTodos() {
        const pool = await sql.connect(config);
        const result = await pool.request()
            .query('SELECT * FROM dbo.Usuarios WHERE Ativo
= 1');
        return result.recordset;
    }

    async criar(dados) {
        const pool = await sql.connect(config);
        const result = await pool.request()
            .input('login', sql.NVarChar, dados.login)
            .input('senha', sql.NVarChar, dados.senha)
            .input('nomeCompleto', sql.NVarChar, dados.nome
Completo)
            .input('email', sql.NVarChar, dados.email)
            .query('INSERT INTO dbo.Usuarios (Login, Senha,
NomeCompleto, Email) OUTPUT INSERTED.* VALUES (@login, @sen
ha, @nomeCompleto, @email)');
        return result.recordset[0];
    }

    async atualizar(id, dados) {
        const pool = await sql.connect(config);
```

```
        const result = await pool.request()
            .input('id', sql.Int, id)
            .input('nomeCompleto', sql.NVarChar, dados.nome
Completo)
            .input('email', sql.NVarChar, dados.email)
            .query('UPDATE dbo.Usuarios SET NomeCompleto =
@nomeCompleto, Email = @email WHERE UsuarioId = @id');
        return result.rowsAffected[0];
    }

    async deletar(id) {
        const pool = await sql.connect(config);
        const result = await pool.request()
            .input('id', sql.Int, id)
            .query('UPDATE dbo.Usuarios SET Ativo = 0 WHERE
UsuarioId = @id');
        return result.rowsAffected[0];
    }
}

module.exports = new UsuarioRepository();
```

4.3 - Criar usuarioController.js

Local: `src/controllers/usuarioController.js`

```
const usuarioRepository = require('../repositories/usuarioR
epository');

class UsuarioController {
    async login(req, res) {
        try {
            const { email, senha } = req.body;
            const usuario = await usuarioRepository.login(e
mail, senha);

            if (!usuario) {
                return res.status(401).json({ erro: 'Usuári
```

```
o ou senha inválidos' });
            }

            res.json(usuario);
        } catch (error) {
            res.status(500).json({ erro: error.message });
        }
    }

    async listarTodos(req, res) {
        try {
            const usuarios = await usuarioRepository.listar
Todos();
            res.json(usuarios);
        } catch (error) {
            res.status(500).json({ erro: error.message });
        }
    }

    async criar(req, res) {
        try {
            const usuario = await usuarioRepository.criar(r
eq.body);
            res.status(201).json(usuario);
        } catch (error) {
            res.status(500).json({ erro: error.message });
        }
    }

    async atualizar(req, res) {
        try {
            const { id } = req.params;
            await usuarioRepository.atualizar(id, req.bod
y);
            res.json({ mensagem: 'Usuário atualizado com su
cesso' });
        } catch (error) {
            res.status(500).json({ erro: error.message });
```

```
        }
    }

    async deletar(req, res) {
        try {
            const { id } = req.params;
            await usuarioRepository.deletar(id);
            res.json({ mensagem: 'Usuário excluído com suce
sso' });
        } catch (error) {
            res.status(500).json({ erro: error.message });
        }
    }
}

module.exports = new UsuarioController();
```

4.4 - Criar usuarioRoutes.js

Local: `src/routes/usuarioRoutes.js`

```
const express = require('express');
const router = express.Router();
const usuarioController = require('../controllers/usuarioCo
ntroller');

router.post('/login', usuarioController.login);
router.get('/', usuarioController.listarTodos);
router.post('/', usuarioController.criar);
router.put('/:id', usuarioController.atualizar);
router.delete('/:id', usuarioController.deletar);

module.exports = router;
```

4.5 - Criar server.js

Local: `server.js` (raiz do projeto)

```
const express = require('express');
const cors = require('cors');
const path = require('path');
require('dotenv').config();

const usuarioRoutes = require('./src/routes/usuarioRoute
s');

const app = express();
const PORT = process.env.PORT || 3000;

app.use(cors());
app.use(express.json());
app.use(express.static('public'));

app.use('/api/usuarios', usuarioRoutes);

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

✅ **Checkpoint:** Backend completo.

FASE 5: FRONTEND (HTML/CSS/JS)

(Arquivos login.html, dashboard.html, CSS e JS conforme o padrão já estabelecido)

✅ **Checkpoint:** Frontend completo.

FASE 6: SINCRONIZAÇÃO GITHUB

```
git init
git add .
git commit -m "AC0: Estrutura inicial completa"
git branch -M main
git remote add origin <https://TOKEN@github.com/buselliroge
```

```
rio/software-product.git>
git push -u origin main
```

✅ **Checkpoint:** Projeto sincronizado.

FASE 7: TESTES

7.1 - Testar Backend (Postman)

```
POST <http://localhost:3000/api/usuarios/login>
Body: { "email": "admin", "senha": "123456" }
```

7.2 - Testar Frontend

```
<http://localhost:3000/pages/login.html>
```

✅ **Checkpoint:** Tudo funcionando.

📊 CHECKLIST COMPLETO

- ☐ Banco criado e testado
- ☐ Pastas estruturadas
- ☐ `.gitignore` criado
- ☐ `package.json` configurado
- ☐ Dependências instaladas
- ☐ `.env` configurado
- ☐ Backend completo
- ☐ Frontend completo
- ☐ GitHub sincronizado
- ☐ Testes realizados

Autor: Buselli Rogerio

Entrega: AC0

Status: Pronto para AC1