

AC0 — Estrutura Inicial do Projeto

AC0 — Estrutura Inicial do Projeto

Projeto: TrocaOleo — Sistema de Gerenciamento de Oficina

Instituição: Faculdade Impacta | **Curso:** ADS

Data: 17/02/2026 | **Versão:** 2.0 (Segurança Implementada)

Autor: Buselli Rogerio

Objetivo do AC0

Estabelecer a estrutura base completa e segura do projeto antes do desenvolvimento das funcionalidades principais, garantindo:

- Organização profissional de pastas e arquivos
 - Configurações de segurança implementadas
 - Ambiente de desenvolvimento robusto
 - Banco de dados estruturado com triggers
 - Backend com autenticação segura
 - Frontend moderno e responsivo
 - Versionamento com Git/GitHub
-

Stack Tecnológica

Backend: Node.js + Express.js + SQL Server + bcrypt

Frontend: HTML5 + CSS3 + JavaScript Vanilla

Banco: SQL Server 2019+ com triggers automáticos

Segurança: Rate limiting + CORS + Hash de senhas + Pool de conexões

Board AC0 — Resumo

Fase	O que é	Status
Fase 1	Estrutura de Pastas	

Fase	O que é	Status
Fase 2	Configuração do Projeto	✓
Fase 3	Banco de Dados (SQL)	✓
Fase 4	Backend Node.js	✓
Fase 5	Frontend HTML	✓
Fase 6	Frontend CSS	✓
Fase 7	Frontend JavaScript	✓
Fase 8	Checklist de Testes	✓
Fase 9	Versionamento GitHub	✓
Fase 10	README	✓

Mapa de Implantação — Ordem Prioritária

FASE 1 — Estrutura de Pastas

Por quê primeiro? Organização é fundamental antes de criar qualquer arquivo. Uma estrutura bem definida facilita manutenção, colaboração e escalabilidade do projeto.

```
# Criar pasta raiz e entrar nela
mkdir C:\software-product
cd C:\software-product

# Criar estrutura de pastas
mkdir sql
mkdir src
mkdir src\config
mkdir src\repositories
mkdir src\controllers
mkdir src\routes
mkdir public
mkdir public\pages
mkdir public\assets
mkdir public\assets\css
```

```
mkdir public\assets\js
mkdir acs
```

Estrutura final:

C:\software-product\ ├─ sql/	← Scripts SQL
├─ src/	← Backend (Node.js)
│ └─ config/	← Pool de conexão com o banco
o	
│ └─ repositories/	← Queries SQL
│ └─ controllers/	← Validações e respostas HTTP
P	
│ └─ routes/	← Rotas da API
├─ public/	← Frontend
│ └─ pages/	← Páginas HTML
│ └─ assets/	
│ │ └─ css/	← Estilos
│ │ └─ js/	← Scripts
├─ acs/	← Token GitHub (NUNCA versionar)
├─ .env	← Variáveis de ambiente (PRIVADO)
├─ .gitignore	← Arquivos ignorados pelo Git
t	
├─ server.js	← Servidor principal
├─ package.json	← Dependências do projeto
├─ package-lock.json	← Lock de versões (gerado pelo npm)
└─ README.md	← Documentação

FASE 2 — Configuração do Projeto

2.1 — `.gitignore`

Por quê primeiro de tudo? O `.gitignore` precisa existir ANTES do primeiro commit para garantir que arquivos sensíveis nunca sejam versionados. Se o

`.env` for enviado ao GitHub com credenciais, o dano é imediato e irreversível mesmo apagando depois.

```
# Dependências do Node
node_modules/

# Configurações sensíveis – NUNCA versionar
.env
acs/Token GitHub.txt

# Logs
*.log
npm-debug.log*

# Sistema operacional
.DS_Store
Thumbs.db

# IDEs
.vscode/
.idea/
*.swp
*.swo
```

2.2 — `.env`

Por quê? Centraliza todas as variáveis de ambiente sensíveis fora do código-fonte. O `database.js` e o `server.js` leem essas variáveis via `process.env`. Nunca versionar este arquivo — ele já está protegido pelo `.gitignore`.

```
# =====
# CONFIGURAÇÕES DO SERVIDOR
# =====
PORT=3000

# =====
# CONFIGURAÇÕES DO BANCO DE DADOS (SQL Server)
```

```
# =====
DB_SERVER=127.0.0.1
DB_DATABASE=SoftwareProduct
DB_USER=sa
DB_PASSWORD=SSBr@194
DB_PORT=1433
```

2.3 — `package.json`

Por quê? Define o projeto Node.js, lista todas as dependências e os scripts de execução. O script `dev` usa nodemon para reiniciar o servidor automaticamente ao salvar qualquer arquivo — essencial durante o desenvolvimento.

```
{
  "name": "software-product",
  "version": "1.0.0",
  "description": "Sistema de gerenciamento acadêmico - Faculdade Impacta",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "repository": {
    "type": "git",
    "url": "git+https://github.com/busellirogerio/software-product.git"
  },
  "author": "Buselli Rogerio",
  "license": "ISC",
  "type": "commonjs",
  "dependencies": {
    "bcrypt": "^6.0.0",
    "cors": "^2.8.6",
    "dotenv": "^17.3.1",
    "express": "^5.2.1",
```

```

    "express-rate-limit": "^8.2.1",
    "mssql": "^12.2.0"
  },
  "devDependencies": {
    "nodemon": "^3.1.11"
  }
}

```

2.4 — package-lock.json

Por quê? Gerado automaticamente pelo `npm install`. Trava as versões exatas de todas as dependências, garantindo que qualquer pessoa que clone o repositório instale exatamente as mesmas versões. Deve ser versionado no Git. Nunca editar manualmente.

```

# Gerado automaticamente ao rodar:
npm install

```

FASE 3 — Banco de Dados

3.1 — sql/CreateSoftwareProduct.sql

Por quê? O banco precisa existir antes de qualquer tabela ou dado. Este script remove o banco anterior se existir e cria um novo limpo, garantindo ambiente consistente a cada execução.

```

-- =====
-- CRIAR BANCO: SoftwareProduct
-- VERSÃO: 1.0 - AC0
-- DATA: 2026-02-15
-- =====

-- Remove banco se existir
IF DB_ID('SoftwareProduct') IS NOT NULL
BEGIN
    ALTER DATABASE SoftwareProduct SET SINGLE_USER WITH ROL

```

```

LBACK IMMEDIATE;
    DROP DATABASE SoftwareProduct;
    PRINT '⚠ Banco anterior removido!';
END
GO

-- Criar novo banco
CREATE DATABASE SoftwareProduct
GO

USE SoftwareProduct;
GO

PRINT '✅ Banco SoftwareProduct criado com sucesso!';
PRINT '✅ Pronto para criar tabelas!';
GO

```

3.2 — `sql/Usuarios.sql`

Por quê? Define a estrutura da tabela `dbo.Usuarios` com colunas de identificação, acesso, dados pessoais, controle lógico (soft delete via campo `Ativo`) e auditoria automática. O trigger `TR_Usuarios_SetDataAtualizacao` atualiza `DataAtualizacao` automaticamente a cada UPDATE sem precisar de lógica extra no backend. Os índices em `Login` e `Ativo` otimizam as queries de autenticação.

```

-- =====
-- BANCO: SoftwareProduct
-- TABELA: dbo.Usuarios
-- VERSÃO: 1.0 - AC0
-- =====

USE SoftwareProduct;
GO

-- =====
-- LIMPEZA – remove objetos existentes
-- =====

```

```

IF OBJECT_ID('dbo.TR_Usuarios_SetDataAtualizacao', 'TR') IS
NOT NULL
    DROP TRIGGER dbo.TR_Usuarios_SetDataAtualizacao;
GO

IF OBJECT_ID('dbo.Usuarios', 'U') IS NOT NULL
    DROP TABLE dbo.Usuarios;
GO

-- =====
-- CRIAÇÃO DA TABELA
-- =====
CREATE TABLE dbo.Usuarios
(
    -- IDENTIFICAÇÃO
    UsuarioId          INT IDENTITY(1,1)  NOT NULL,

    -- DADOS DE ACESSO
    Login              NVARCHAR(100)      NOT NULL,
    Senha              NVARCHAR(255)      NOT NULL,

    -- DADOS PESSOAIS
    NomeCompleto       NVARCHAR(120)      NOT NULL,
    Email              NVARCHAR(254)      NULL,

    -- CONTROLE LÓGICO (soft delete)
    Ativo              BIT                 NOT NULL
        CONSTRAINT DF_Usuarios_Ativo DEFAULT (1),

    -- AUDITORIA
    DataCriacao        DATETIME2(0)       NOT NULL
        CONSTRAINT DF_Usuarios_DataCriacao DEFAULT (SYSDATE
TIME()),
    DataAtualizacao    DATETIME2(0)       NOT NULL
        CONSTRAINT DF_Usuarios_DataAtualizacao DEFAULT (SYS
DATETIME()),

    -- CONSTRAINTS

```

```

        CONSTRAINT PK_Usuarios
            PRIMARY KEY CLUSTERED (UsuarioId),
        CONSTRAINT UQ_Usuarios_Login
            UNIQUE (Login),
        CONSTRAINT CK_Usuarios_Email_Formato
            CHECK (Email IS NULL OR Email LIKE '%_@_%._%')
    );
GO

-- =====
-- ÍNDICES – otimizam queries de autenticação
-- =====
CREATE NONCLUSTERED INDEX IX_Usuarios_Login
    ON dbo.Usuarios (Login)
    WHERE Ativo = 1;
GO

CREATE NONCLUSTERED INDEX IX_Usuarios_Ativo
    ON dbo.Usuarios (Ativo)
    INCLUDE (UsuarioId, Login, NomeCompleto);
GO

-- =====
-- TRIGGER – atualiza DataAtualizacao automaticamente
-- =====
CREATE TRIGGER dbo.TR_Usuarios_SetDataAtualizacao
    ON dbo.Usuarios
    AFTER UPDATE
    AS
    BEGIN
        SET NOCOUNT ON;
        UPDATE u
            SET DataAtualizacao = SYSDATETIME()
        FROM dbo.Usuarios u
        INNER JOIN inserted i ON i.UsuarioId = u.UsuarioId;
    END;
GO

```

```
PRINT '✅ Tabela dbo.Usuarios criada com sucesso!';
GO
```

3.3 — `sql/TesteUsuarios.sql`

Por quê? Garante que a tabela funciona corretamente antes de subir o backend. Executa o ciclo completo: zera dados, insere, atualiza e faz soft delete. Atenção: a senha aqui é texto puro apenas para teste direto no banco. No sistema real o backend sempre faz o hash via bcrypt antes de gravar.

```
USE SoftwareProduct;
GO

-- =====
-- 1) ZERAR TABELA E REINICIAR CONTADOR
-- =====
DELETE FROM dbo.Usuarios;
DBCC CHECKIDENT ('dbo.Usuarios', RESEED, 0);
GO

-- =====
-- 2) VERIFICAR SE ZEROU
-- =====
SELECT * FROM dbo.Usuarios;
GO

-- =====
-- 3) INSERIR REGISTRO DE TESTE
-- =====
INSERT INTO dbo.Usuarios (Login, Senha, NomeCompleto, Email, Ativo)
VALUES ('admin', 'Senha@123', 'Teste01', 'teste01@email.com', 1);
GO

-- =====
-- 4) ATUALIZAR (verifica trigger DataAtualizacao)
```

```
-- =====
UPDATE dbo.Usuarios
SET NomeCompleto = 'Administrador Atualizado'
WHERE Login = 'admin';
GO

-- =====
-- 5) SOFT DELETE (Ativo = 0, não apaga o registro)
-- =====
UPDATE dbo.Usuarios
SET Ativo = 0
WHERE Login = 'admin';
GO
```

FASE 4 — Backend

4.1 — `src/config/database.js`

Por quê primeiro no backend? Toda a camada de dados depende da conexão com o banco. O pool mantém entre 2 e 10 conexões simultâneas reutilizáveis, evitando abrir e fechar conexão a cada requisição. Valida variáveis de ambiente obrigatórias ao iniciar — se faltar alguma, o servidor não sobe.

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

/* =====
   VALIDAÇÃO DE VARIÁVEIS
===== */
const requiredEnvVars = ['DB_SERVER', 'DB_DATABASE', 'DB_US
ER', 'DB_PASSWORD'];
const missingVars = requiredEnvVars.filter((varName) => !pr
ocess.env[varName]);

if (missingVars.length > 0) {
```

```

    console.error('❌ Variáveis de ambiente obrigatórias não
definidas:', missingVars);
    process.exit(1);
}

/* =====
CONFIGURAÇÃO DO POOL
===== */
const config = {
  server: process.env.DB_SERVER,
  database: process.env.DB_DATABASE,
  user: process.env.DB_USER,
  password: process.env.DB_PASSWORD,
  port: parseInt(process.env.DB_PORT, 10) || 1433,
  pool: {
    max: 10, // máximo de conexões simultâneas
    min: 2, // mínimo mantido em standby
    idleTimeoutMillis: 30000,
  },
  options: {
    encrypt: false,
    trustServerCertificate: true,
    connectTimeout: 60000,
    requestTimeout: 30000,
    enableArithAbort: true,
  },
};

/* =====
GERENCIAMENTO DO POOL
===== */
let globalPool = null;

const getPool = async () => {
  try {
    if (!globalPool) {
      console.log('🔧 Criando pool de conexões com SQL Serv
er...');

```

```

    globalPool = await sql.connect(config);

    globalPool.on('error', (err) => {
        console.error('❌ Erro na conexão SQL Server:', err);
        globalPool = null;
    });

    console.log('✅ Pool de conexões SQL Server criado com sucesso');
    }
    return globalPool;
} catch (error) {
    console.error('❌ Erro ao criar pool de conexões:', error);
    globalPool = null;
    throw error;
}
};

/* =====
UTILITÁRIOS
===== */
const testConnection = async () => {
    try {
        const pool = await getPool();
        await pool.request().query('SELECT 1 as test');
        console.log('✅ Teste de conectividade SQL Server: OK');
        return true;
    } catch (error) {
        console.error('❌ Teste de conectividade SQL Server: FALHOU');
        return false;
    }
};

const closePool = async () => {

```

```

    if (globalPool) {
      await globalPool.close();
      globalPool = null;
    }
  };

  module.exports = { config, getPool, testConnection, closePool, sql };

```

4.2 — `src/repositories/usuarioRepository.js`

Por quê? Camada exclusiva de acesso ao banco. Toda query SQL fica centralizada aqui — nenhum outro arquivo faz query diretamente. Senhas nunca são gravadas em texto puro: bcrypt gera o hash com salt de 10 rounds antes de qualquer INSERT ou UPDATE. O método `login` usa `bcrypt.compare` para validar sem descriptografar. O `resetSenha` gera nova senha aleatória, grava o hash e retorna apenas o protocolo.

```

// src/repositories/usuarioRepository.js
const { getPool, sql } = require('../config/database');
const bcrypt = require('bcrypt');

class UsuarioRepository {

  /** =====
   LOGIN
   ===== */
  async login(email, senha) {
    const pool = await getPool();

    const result = await pool
      .request()
      .input('email', sql.NVarChar, email)
      .query(`
        SELECT *
        FROM dbo.Usuarios
        WHERE (Email = @email OR Login = @email)
        AND Ativo = 1
      `);
  }
}

```

```

    `);

    const usuario = result.recordset[0];
    if (!usuario) return null;

    // bcrypt.compare – valida sem descriptografar
    const senhaValida = await bcrypt.compare(senha, usuario.Senha);
    return senhaValida ? usuario : null;
}

/* =====
   LISTAR TODOS
   ===== */
async listarTodos() {
    const pool = await getPool();

    const result = await pool
        .request()
        .query(`
            SELECT UsuarioId, Login, NomeCompleto, Email, Ativo, DataCriacao
            FROM dbo.Usuarios
            WHERE Ativo = 1
        `);

    return result.recordset;
}

/* =====
   CRIAR
   ===== */
async criar(dados) {
    const pool = await getPool();

    // Hash da senha antes de gravar
    const senhaHash = await bcrypt.hash(dados.senha, 10);

```

```

const result = await pool
  .request()
  .input('login', sql.NVarChar, dados.login)
  .input('senha', sql.NVarChar, senhaHash)
  .input('nomeCompleto', sql.NVarChar, dados.nomeCompleto)
  .input('email', sql.NVarChar, dados.email)
  .query(`
    INSERT INTO dbo.Usuarios (Login, Senha, NomeCompleto, Email, Ativo)
    OUTPUT
      INSERTED.UsuarioId,
      INSERTED.Login,
      INSERTED.NomeCompleto,
      INSERTED.Email,
      INSERTED.Ativo,
      INSERTED.DataCriacao
    VALUES (@login, @senha, @nomeCompleto, @email, 1)
  `);

return result.recordset[0];
}

/* =====
   ATUALIZAR
===== */
async atualizar(id, dados) {
  const pool = await getPool();

  const senhaHash = await bcrypt.hash(dados.senha, 10);

  const result = await pool
    .request()
    .input('id', sql.Int, id)
    .input('nomeCompleto', sql.NVarChar, dados.nomeCompleto)
    .input('email', sql.NVarChar, dados.email)
    .input('senha', sql.NVarChar, senhaHash)

```

```

        .query(`
            UPDATE dbo.Usuarios
            SET NomeCompleto = @nomeCompleto,
                Email        = @email,
                Senha         = @senha
            WHERE UsuarioId = @id
        `);

```

```

    return result.rowsAffected[0];
}

```

```

/* =====
   DELETAR (soft delete)
===== */

```

```

async deletar(id) {
    const pool = await getPool();

```

```

    const result = await pool
        .request()
        .input('id', sql.Int, id)
        .query(`
            UPDATE dbo.Usuarios
            SET Ativo = 0
            WHERE UsuarioId = @id
        `);

```

```

    return result.rowsAffected[0];
}

```

```

/* =====
   RESET DE SENHA
===== */

```

```

async resetSenha(email) {
    const pool = await getPool();

```

```

    // Verifica se usuário existe
    const user = await pool
        .request()

```

```

        .input('email', sql.NVarChar, email)
        .query(`
            SELECT UsuarioId
            FROM dbo.Usuarios
            WHERE Email = @email AND Ativo = 1
        `);

    if (!user.recordset[0]) return null;

    // Gera nova senha aleatória e faz hash
    const novaSenha = Math.random().toString(36).slice(-8);
    const senhaHash = await bcrypt.hash(novaSenha, 10);

    await pool
        .request()
        .input('email', sql.NVarChar, email)
        .input('senha', sql.NVarChar, senhaHash)
        .query(`
            UPDATE dbo.Usuarios
            SET Senha = @senha
            WHERE Email = @email
        `);

    // Gera protocolo de atendimento
    const ano = new Date().getFullYear();
    const protocolo = `TI-${ano}-${Math.floor(100000 + Math
h.random() * 900000)}`;

    console.log('RESET SENHA => Email:', email, '| Protocol
o:', protocolo);

    return { protocolo };
}
}

module.exports = new UsuarioRepository();

```

4.3 — `src/controllers/usuarioController.js`

Por quê? Camada intermediária entre as rotas e o repository. Valida dados recebidos, trata erros e formata respostas. Remove campos sensíveis como **Senha** antes de responder ao cliente. Retorna status HTTP semânticos: 400 (dados inválidos), 401 (não autorizado), 404 (não encontrado), 409 (duplicidade), 500 (erro interno).

```
// src/controllers/usuarioController.js
const usuarioRepository = require('../repositories/usuarioRepository');

class UsuarioController {

  /* =====
  LOGIN
  ===== */
  async login(req, res) {
    try {
      const { email, senha } = req.body;

      if (!email || !senha) {
        return res.status(400).json({ erro: 'Email e senha são obrigatórios' });
      }

      const usuario = await usuarioRepository.login(email.trim(), senha);

      if (!usuario) {
        return res.status(401).json({ erro: 'Usuário ou senha inválidos' });
      }

      // Remove campo Senha antes de retornar
      const { Senha, ...usuarioSemSenha } = usuario;
      res.json(usuarioSemSenha);
    } catch (error) {
      console.error('Erro no login:', error);
    }
  }
}
```

```

        res.status(500).json({ erro: 'Erro interno do servido
r' });
    }
}

/* =====
   RESET DE SENHA
   ===== */
async resetSenha(req, res) {
    try {
        const { email } = req.body;

        if (!email) {
            return res.status(400).json({ erro: 'Email é obriga
tório' });
        }

        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!emailRegex.test(email)) {
            return res.status(400).json({ erro: 'Formato de ema
il inválido' });
        }

        const resultado = await usuarioRepository.resetSenha
(email.trim());

        if (!resultado) {
            return res.status(404).json({ erro: 'Usuário não en
contrado' });
        }

        res.json({
            mensagem: 'Reset realizado com sucesso',
            protocolo: resultado.protocolo,
        });

    } catch (error) {
        console.error('Erro no reset:', error);
    }
}

```

```

        res.status(500).json({ erro: 'Erro interno do servido
r' });
    }
}

/* =====
LISTAR TODOS
===== */
async listarTodos(req, res) {
    try {
        const usuarios = await usuarioRepository.listarTodos
();
        res.json(usuarios);
    } catch (error) {
        console.error('Erro ao listar:', error);
        res.status(500).json({ erro: 'Erro interno do servido
r' });
    }
}

/* =====
CRIAR
===== */
async criar(req, res) {
    try {
        const { login, senha, nomeCompleto, email } = req.bod
y;

        if (!login || !senha || !nomeCompleto || !email) {
            return res.status(400).json({ erro: 'Login, senha,
nome completo e email são obrigatórios' });
        }

        if (senha.length < 6) {
            return res.status(400).json({ erro: 'Senha deve ter
pelo menos 6 caracteres' });
        }
    }
}

```

```

    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    if (!emailRegex.test(email)) {
        return res.status(400).json({ erro: 'Formato de email inválido' });
    }

    const dadosLimpos = {
        login: login.trim(),
        senha,
        nomeCompleto: nomeCompleto.trim(),
        email: email.trim().toLowerCase(),
    };

    const usuario = await usuarioRepository.criar(dadosLimpos);
    res.status(201).json(usuario);

} catch (error) {
    if (error.message.includes('UNIQUE')) {
        return res.status(409).json({ erro: 'Login ou email já existem' });
    }
    console.error('Erro ao criar:', error);
    res.status(500).json({ erro: 'Erro interno do servidor' });
}

}

/* =====
   ATUALIZAR
   ===== */
async atualizar(req, res) {
    try {
        const { id } = req.params;
        const { nomeCompleto, email, senha } = req.body;

        if (!id || isNaN(id)) {
            return res.status(400).json({ erro: 'ID inválido'

```

```

    });

    }

    if (!nomeCompleto || !email || !senha) {
        return res.status(400).json({ erro: 'Nome completo,
email e senha são obrigatórios' });
    }

    if (senha.length < 6) {
        return res.status(400).json({ erro: 'Senha deve ter
peelo menos 6 caracteres' });
    }

    const resultado = await usuarioRepository.atualizar(id, {
        nomeCompleto: nomeCompleto.trim(),
        email: email.trim().toLowerCase(),
        senha,
    });

    if (resultado === 0) {
        return res.status(404).json({ erro: 'Usuário não en
contrado' });
    }

    res.json({ mensagem: 'Usuário atualizado com sucesso'
});

    } catch (error) {
        console.error('Erro ao atualizar:', error);
        res.status(500).json({ erro: 'Erro interno do servido
r' });
    }
}

/* =====
DELETAR (soft delete)
===== */

```

```

async deletar(req, res) {
  try {
    const { id } = req.params;

    if (!id || isNaN(id)) {
      return res.status(400).json({ erro: 'ID inválido'
});
    }

    const resultado = await usuarioRepository.deletar(i
d);

    if (resultado === 0) {
      return res.status(404).json({ erro: 'Usuário não en
contrado' });
    }

    res.json({ mensagem: 'Usuário excluído com sucesso'
});

  } catch (error) {
    console.error('Erro ao deletar:', error);
    res.status(500).json({ erro: 'Erro interno do servido
r' });
  }
}

module.exports = new UsuarioController();

```

4.4 — src/routes/usuarioRoutes.js

Por quê? Define todas as rotas da API com rate limiting diferenciado por criticidade. Login aceita apenas 5 tentativas por IP a cada 15 minutos (proteção contra força bruta), reset aceita 3 por hora, demais rotas 100 por 15 minutos. O middleware `validateJSON` rejeita requisições POST/PUT sem body antes de chegar ao controller.

```
// src/routes/usuarioRoutes.js
const express = require('express');
const rateLimit = require('express-rate-limit');
const router = express.Router();
const usuarioController = require('../controllers/usuarioController');

/* =====
   RATE LIMITERS
   ===== */

// Login – 5 tentativas por IP a cada 15 min (anti força bruta)
const loginLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 5,
  message: { erro: 'Muitas tentativas de login. Tente novamente em 15 minutos.' },
  standardHeaders: true,
  legacyHeaders: false,
});

// Reset de senha – 3 solicitações por hora
const resetLimiter = rateLimit({
  windowMs: 60 * 60 * 1000,
  max: 3,
  message: { erro: 'Muitas solicitações de reset. Tente novamente em 1 hora.' },
  standardHeaders: true,
  legacyHeaders: false,
});

// Rotas gerais – 100 requisições por 15 min
const generalLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 100,
  message: { erro: 'Muitas requisições. Tente novamente em 15 minutos.' },
});
```

```

    standardHeaders: true,
    legacyHeaders: false,
  });

  /* =====
    VALIDAÇÃO DE BODY
    ===== */
  const validateJSON = (req, res, next) => {
    if (['POST', 'PUT'].includes(req.method)) {
      if (!req.body || Object.keys(req.body).length === 0) {
        return res.status(400).json({ erro: 'Dados obrigatórios não fornecidos' });
      }
    }
    next();
  };

  /* =====
    ROTAS PÚBLICAS
    ===== */
  router.post('/login', loginLimiter, validateJSON, usuarioController.login);
  router.post('/reset-senha', resetLimiter, validateJSON, usuarioController.resetSenha);

  /* =====
    ROTAS PROTEGIDAS
    ===== */
  router.get('/', generalLimiter, usuarioController.listarTodos);
  router.post('/', generalLimiter, validateJSON, usuarioController.criar);
  router.put('/:id', generalLimiter, validateJSON, usuarioController.atualizar);
  router.delete('/:id', generalLimiter, usuarioController.deletar);

  module.exports = router;

```

4.5 — `server.js`

Por quê último no backend? É o ponto de entrada que une tudo. Configura Express, CORS (apenas origens autorizadas), rate limiting global, headers de segurança HTTP, serve os arquivos estáticos do frontend e registra as rotas da API. Rotas explícitas para `/` e `/pages/dashboard.html` com URLs absolutas evitam conflito com Live Server na porta 5500.

```
// server.js
const express = require('express');
const cors = require('cors');
const rateLimit = require('express-rate-limit');
const path = require('path');
require('dotenv').config();

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

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

/* =====
   RATE LIMIT GLOBAL
   ===== */
const globalLimiter = rateLimit({
  windowMs: 15 * 60 * 1000,
  max: 1000,
  message: { erro: 'Servidor sobrecarregado. Tente novamente.' },
  standardHeaders: true,
  legacyHeaders: false,
});

/* =====
   CORS
   ===== */
const corsOptions = {
  origin: [
```

```

    'http://localhost:3000',
    'http://127.0.0.1:3000',
    'http://localhost:5500',
    'http://127.0.0.1:5500',
  ],
  methods: ['GET', 'POST', 'PUT', 'DELETE'],
  allowedHeaders: ['Content-Type', 'Authorization'],
};

/* =====
   SECURITY HEADERS
===== */
app.use((req, res, next) => {
  res.setHeader('X-Content-Type-Options', 'nosniff');
  res.setHeader('X-Frame-Options', 'DENY');
  res.setHeader('X-XSS-Protection', '1; mode=block');
  res.removeHeader('X-Powered-By');
  next();
});

app.use(globalLimiter);
app.use(cors(corsOptions));

/* =====
   STATIC FILES
===== */
app.use(express.static(path.join(__dirname, 'public')));

/* =====
   API – JSON E VALIDAÇÃO
===== */
app.use('/api', express.json({ limit: '10mb' }));

app.use('/api', (req, res, next) => {
  if (['POST', 'PUT', 'PATCH'].includes(req.method) && !req.is('application/json')) {
    return res.status(415).json({ erro: 'Content-Type deve ser application/json' });
  }
  next();
});

```

```

    }
    next();
  });

  /* =====
  LOG DE REQUISIÇÕES
  ===== */
  app.use((req, res, next) => {
    console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
    next();
  });

  /* =====
  ROTAS API
  ===== */
  app.use('/api/usuarios', usuarioRoutes);

  /* =====
  ROTAS FRONT
  ===== */
  app.get('/', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'pages', 'login.html'));
  });

  app.get('/pages/dashboard.html', (req, res) => {
    res.sendFile(path.join(__dirname, 'public', 'pages', 'dashboard.html'));
  });

  /* =====
  404
  ===== */
  app.use((req, res) => {
    if (req.url.startsWith('/api')) {
      res.status(404).json({ erro: 'Endpoint não encontrado' });
    }
  });

```

```

    } else {
      res.status(404).send('404 - Página não encontrada');
    }
  });

  /* =====
  START
  ===== */
  app.listen(PORT, () => {
    console.log('=====');
    console.log(`Servidor rodando: http://127.0.0.1:${PORT}`);
    console.log(`Login:      http://127.0.0.1:${PORT}/pages/login.html`);
    console.log(`Dashboard: http://127.0.0.1:${PORT}/pages/dashboard.html`);
    console.log('=====');
  });

```

FASE 5 — Frontend HTML

5.1 — `public/pages/login.html`

Por quê HTML primeiro? Você estrutura o esqueleto antes de estilizar. O HTML define os elementos que o CSS vai decorar e o JS vai manipular. Carrega `style.css` → `login.css` para estilos e `config.js` → `login.js` para lógica. Inclui modal de recuperação de senha com exibição do protocolo gerado pelo backend.

```

<!doctype html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Login - Software Product</title>

```

```

    <!-- CSS – style.css primeiro (variáveis globais) -->
    <link rel="stylesheet" href="../../assets/css/style.css" /
>
    <link rel="stylesheet" href="../../assets/css/login.css" /
>
</head>
<body>

    <!-- =====
        CONTAINER DE LOGIN
    ===== -->
    <div class="login-container">
        <div class="login-box">

            <!-- HEADER -->
            <div class="login-header">
                <h1>🛢 TROCA DE ÓLEO</h1>
                <p>Sistema de Gerenciamento Oficina</p>
            </div>

            <!-- FORMULÁRIO DE LOGIN -->
            <form id="formLogin" class="login-form">

                <div class="form-group">
                    <label for="email">Usuário</label>
                    <input
                        type="text"
                        id="email"
                        name="email"
                        required
                        autocomplete="username"
                    />
                </div>

                <div class="form-group">
                    <label for="senha">Senha</label>
                    <div class="password-container">
                        <input

```

```

        type="password"
        id="senha"
        name="senha"
        required
        autocomplete="current-password"
    />
    <button
        type="button"
        id="toggleSenha"
        class="toggle-password"
        aria-label="Mostrar senha"
    ></button>
</div>
</div>

<button type="submit" class="btn-login">Entrar</b
utton>

<div class="form-footer">
    <a href="#" id="linkEsqueciSenha" class="link-e
squeci">
        Esqueci minha senha
    </a>
</div>

</form>
</div>
</div>

<!-- =====
      MODAL – RECUPERAÇÃO DE SENHA
      ===== -->
<div id="modalEsqueciSenha" class="modal">
    <div class="modal-content">

        <!-- HEADER DO MODAL -->
        <div class="modal-header">
            <h3>Recuperar Senha</h3>

```

```

        <button type="button" class="modal-close" id="closeModal">&times;</button>
    </div>

    <!-- FORMULÁRIO DE RESET -->
    <form id="formResetSenha" class="modal-form">
        <div class="form-group">
            <label for="emailReset">Digite seu email:</label>

            <input
                type="email"
                id="emailReset"
                name="email"
                required
                placeholder="seu@email.com"
            />
        </div>
        <div class="modal-buttons">
            <button type="button" id="btnCancelar" class="btn-cancelar">Cancelar</button>
            <button type="submit" class="btn-enviar">Enviar Solicitação</button>
        </div>
    </form>

    <!-- TELA DE SUCESSO (exibida após reset) -->
    <div id="resetSucesso" class="reset-success" style="display: none">
        <div class="success-icon">✔</div>
        <h4>Solicitação Enviada!</h4>
        <p>Sua nova senha foi gerada com sucesso.</p>
        <div class="protocolo-info">
            <strong>Protocolo: <span id="numeroProtocolo">
        </span></strong>
            <p>Anote este número para consultas futuras</p>
        </div>
        <button type="button" id="btnFecharSucesso" class="btn-fechar">Fechar</button>
    </div>

```

```

    </div>

    </div>
</div>

<!-- =====
      SCRIPTS
===== -->
<script src="../../assets/js/config.js"></script>
<script src="../../assets/js/login.js"></script>

<!-- Lógica do modal de recuperação de senha -->
<script>
    document.addEventListener('DOMContentLoaded', () => {
        const modal          = document.getElementById('modalEsqueciSenha');
        const linkEsqueci     = document.getElementById('linkEsqueciSenha');
        const closeModal      = document.getElementById('closeModal');
        const btnCancelar     = document.getElementById('btnCancelar');
        const formReset        = document.getElementById('formResetSenha');
        const resetSucesso     = document.getElementById('resetSucesso');
        const btnFecharSucesso = document.getElementById('btnFecharSucesso');

        // Abrir modal
        linkEsqueci.addEventListener('click', (e) => {
            e.preventDefault();
            modal.style.display = 'block';
            document.getElementById('emailReset').focus();
        });

        // Fechar modal
        const fecharModal = () => {

```

```

        modal.style.display = 'none';
        formReset.style.display = 'block';
        resetSucesso.style.display = 'none';
        formReset.reset();
    };

    closeModal.addEventListener('click', fecharModal);
    btnCancelar.addEventListener('click', fecharModal);
    btnFecharSucesso.addEventListener('click', fecharMo
dal);

    modal.addEventListener('click', (e) => {
        if (e.target === modal) fecharModal();
    });

    // Enviar solicitação de reset
    formReset.addEventListener('submit', async (e) => {
        e.preventDefault();
        const email = document.getElementById('emailR
eset').value.trim();
        const btnEnviar = formReset.querySelector('.btn-e
nviar');

        if (!email) { alert('Digite seu email'); return;
    }

    btnEnviar.disabled = true;
    btnEnviar.textContent = 'Enviando...';

    try {
        const response = await apiRequest('/usuarios/re
set-senha', {
            method: 'POST',
            body: { email },
        });
        document.getElementById('numeroProtocolo').text
Content = response.protocolo;
        formReset.style.display = 'none';
        resetSucesso.style.display = 'block';
    }

```

```

    } catch (error) {
        alert(error.message || 'Erro ao enviar solicita
ção');
    } finally {
        btnEnviar.disabled = false;
        btnEnviar.textContent = 'Enviar Solicitação';
    }
    });
});
</script>

</body>
</html>

```

5.2 — public/pages/dashboard.html

Por quê? Painel principal do sistema. Carrega `style.css` → `dashboard.css` para estilos e `config.js` → `auth.js` → `dashboard.js` nessa ordem obrigatória: auth precisa do config já carregado, e dashboard precisa do auth já ter verificado a sessão.

```

<!doctype html>
<html lang="pt-BR">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, init
ial-scale=1.0" />
    <title>Software Product - Dashboard</title>

    <!-- CSS — style.css primeiro (variáveis globais) -->
    <link rel="stylesheet" href="../assets/css/style.css" /
  >
    <link rel="stylesheet" href="../assets/css/dashboard.cs
s" />
  </head>
  <body>

    <!-- =====

```

HEADER

```
===== -->
<header class="header">
  <div class="header-content">
    <h1>🛢️ TROCA DE ÓLEO</h1>
    <div class="user-info">
      <span>Bem-vindo, <strong id="nomeUsuario">Usuário
</strong></span>
      <button id="btnSair" class="btn-sair">Sair</button>
    </div>
  </div>
</header>

<!-- =====
      CONTEÚDO PRINCIPAL
===== -->
<main class="main-content">
  <h2>Dashboard</h2>

  <!-- CARDS DE ESTATÍSTICAS -->
  <div class="stats-container">
    <div class="stat-card">
      <div class="stat-icon">👤</div>
      <h3 id="totalUsuarios">0</h3>
      <p>CLIENTES</p>
    </div>
    <div class="stat-card">
      <div class="stat-icon">🚗</div>
      <h3 id="totalModulos">1</h3>
      <p>VEÍCULOS</p>
    </div>
    <div class="stat-card">
      <div class="stat-icon">✅</div>
      <h3 id="totalSistema">100%</h3>
      <p>SERVIÇOS</p>
    </div>
  </div>
```

```

<!-- ACESSO RÁPIDO -->
<div class="quick-access">
  <h3>Acesso Rápido</h3>
  <div class="quick-buttons">
    <a href="usuarios.html" class="quick-btn">
      <span>👤</span>
      <p>CADASTRO CLIENTES</p>
    </a>
    <a href="clientes.html" class="quick-btn">
      <span>👤</span>
      <p>CADASTRO VEÍCULOS</p>
    </a>
    <a href="relatorios.html" class="quick-btn">
      <span>📋</span>
      <p>HISTÓRICO SERVIÇOS</p>
    </a>
    <a href="configuracoes.html" class="quick-btn">
      <span>⚙️</span>
      <p>EVENTOS FUTUROS</p>
    </a>
  </div>
</div>
</main>

<!-- =====
      SCRIPTS (ordem obrigatória)
===== -->
<script src="../assets/js/config.js"></script>
<script src="../assets/js/auth.js"></script>
<script src="../assets/js/dashboard.js"></script>

</body>
</html>

```

FASE 6 — Frontend CSS

6.1 — public/assets/css/style.css

Por quê primeiro entre os CSS? Define as variáveis CSS no `:root` usadas por todos os outros arquivos de estilo. Sem ele, `login.css` e `dashboard.css` ficam sem `--primary-color`, `--spacing-lg`, `--border-radius` etc. Contém também reset global, tipografia base e animações reutilizáveis.

```
/* =====
   RESET GLOBAL
===== */
* {
  margin: 0;
  padding: 0;
  box-sizing: border-box;
}

html {
  font-size: 16px;
  scroll-behavior: smooth;
}

body {
  font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
  line-height: 1.6;
  color: #ffffff;
  background: #1e2a45;
  min-height: 100vh;
  overflow-x: hidden;
}

/* =====
   VARIÁVEIS GLOBAIS
===== */
:root {
  /* Cores */
  --primary-color: #3498db;
  --primary-dark: #2980b9;
```

```

--secondary-color: #2c3e50;
--success-color: #2ecc71;
--warning-color: #f39c12;
--danger-color: #e74c3c;
--white: #ffffff;
--light-gray: #ecf0f1;
--gray: #bdc3c7;
--dark-gray: #7f8c8d;

/* Sombras */
--shadow-light: 0 2px 4px rgba(0, 0, 0, 0.2);
--shadow-medium: 0 4px 8px rgba(0, 0, 0, 0.3);

/* Bordas */
--border-radius: 8px;
--border-radius-large: 12px;

/* Transições */
--transition-fast: 0.2s ease;
--transition-normal: 0.3s ease;

/* Espaçamentos */
--spacing-sm: 8px;
--spacing-md: 16px;
--spacing-lg: 24px;
--spacing-xl: 32px;
--spacing-xxl: 48px;
}

/* =====
   TIPOGRAFIA
   ===== */
h1, h2, h3, h4, h5, h6 {
  font-weight: 600;
  line-height: 1.3;
  margin-bottom: var(--spacing-md);
}

```

```

/* =====
FORMULÁRIOS
===== */

.form-group {
  margin-bottom: var(--spacing-lg);
}

label {
  display: block;
  font-weight: 500;
  margin-bottom: var(--spacing-sm);
  color: #a0aec0;
  font-size: 0.9rem;
}

input,
select,
textarea {
  width: 100%;
  padding: var(--spacing-md);
  border: 1px solid #2d3748;
  border-radius: var(--border-radius);
  font-size: 1rem;
  font-family: inherit;
  background: #111827;
  color: #ffffff;
  transition: border-color var(--transition-fast),
    box-shadow var(--transition-fast);
}

input:focus,
select:focus,
textarea:focus {
  outline: none;
  border-color: var(--primary-color);
  box-shadow: 0 0 0 3px rgba(52, 152, 219, 0.15);
}

```

```

/* =====
   BOTÃO BASE
===== */
.btn {
  display: inline-flex;
  align-items: center;
  justify-content: center;
  padding: var(--spacing-md) var(--spacing-lg);
  border: none;
  border-radius: var(--border-radius);
  font-size: 1rem;
  font-weight: 600;
  cursor: pointer;
  transition: all var(--transition-fast);
  min-height: 44px;
}

.btn:disabled {
  opacity: 0.6;
  cursor: not-allowed;
}

/* =====
   ANIMAÇÕES
===== */
@keyframes fadeIn {
  from { opacity: 0; }
  to   { opacity: 1; }
}

@keyframes slideIn {
  from { transform: translateY(-20px); opacity: 0; }
  to   { transform: translateY(0);      opacity: 1; }
}

@keyframes spin {
  0%   { transform: rotate(0deg); }

```

```
100% { transform: rotate(360deg); }  
}
```

6.2 — public/assets/css/login.css

Por quê? Estilos exclusivos da tela de login: container centralizado na viewport, caixa com fundo escuro #1e2a45, campo de senha com botão toggle, mensagens de erro/sucesso e modal de recuperação com header azul e corpo branco para contraste. Responsivo com breakpoints em 768px e 480px.

```
/* =====  
CONTAINER DE LOGIN  
===== */  
.login-container {  
  display: flex;  
  align-items: center;  
  justify-content: center;  
  min-height: 100vh;  
  background: #1e2a45;  
  padding: var(--spacing-lg);  
}  
  
.login-box {  
  background: #1e2a45;  
  border-radius: var(--border-radius-large);  
  box-shadow: 0 20px 60px rgba(0, 0, 0, 0.4);  
  padding: var(--spacing-xxl) var(--spacing-xl);  
  width: 100%;  
  max-width: 480px;  
  animation: slideIn 0.4s ease-out;  
}  
  
/* =====  
HEADER DO LOGIN  
===== */  
.login-header {  
  text-align: center;
```

```

    margin-bottom: var(--spacing-xl);
}

.login-header h1 {
    font-size: 2rem;
    color: #ffffff;
    margin-bottom: var(--spacing-sm);
    font-weight: 700;
    letter-spacing: 1px;
}

.login-header p {
    color: #a0aec0;
    font-size: 0.85rem;
    margin: 0;
    letter-spacing: 2px;
    text-transform: uppercase;
}

/* =====
   CAMPOS DO FORMULÁRIO
===== */

.form-group {
    margin-bottom: var(--spacing-lg);
}

.form-group label {
    color: #a0aec0;
    font-size: 0.9rem;
    margin-bottom: 8px;
}

.form-group input {
    background: #111827;
    border: 1px solid #2d3748;
    color: #ffffff;
    border-radius: var(--border-radius);
    padding: 14px 16px;
}

```

```

    font-size: 1rem;
}

.form-group input:focus {
    border-color: var(--primary-color);
    box-shadow: 0 0 0 3px rgba(52, 152, 219, 0.15);
}

/* =====
    CAMPO DE SENHA COM TOGGLE
===== */
.password-container {
    position: relative;
    display: flex;
    align-items: center;
}

.password-container input {
    width: 100%;
    padding-right: 48px;
}

.toggle-password {
    position: absolute;
    right: 12px;
    background: transparent;
    border: none;
    cursor: pointer;
    padding: 4px;
    color: #a0aec0;
    display: flex;
    align-items: center;
    justify-content: center;
    transition: color var(--transition-fast);
}

.toggle-password:hover {
    color: var(--primary-color);
}

```

```

}

/* =====
   BOTÃO DE LOGIN
===== */
.btn-login {
  width: 100%;
  background: var(--primary-color);
  color: var(--white);
  border: none;
  padding: 16px;
  border-radius: var(--border-radius);
  font-size: 1rem;
  font-weight: 700;
  cursor: pointer;
  transition: all var(--transition-normal);
  margin: var(--spacing-lg) 0;
  letter-spacing: 1px;
  text-transform: uppercase;
}

.btn-login:hover:not(:disabled) {
  background: var(--primary-dark);
  transform: translateY(-2px);
  box-shadow: 0 8px 20px rgba(52, 152, 219, 0.4);
}

.btn-login:disabled {
  opacity: 0.6;
  cursor: not-allowed;
}

/* =====
   LINK ESQUECI SENHA
===== */
.form-footer {
  text-align: center;
  margin-top: var(--spacing-md);
}

```

```

}

.link-esqueci {
  display: block;
  width: 100%;
  padding: 12px;
  border: 1px solid var(--primary-color);
  border-radius: var(--border-radius);
  color: var(--primary-color);
  font-size: 0.9rem;
  text-decoration: none;
  text-align: center;
  transition: all var(--transition-fast);
  background: transparent;
  cursor: pointer;
}

.link-esqueci:hover {
  background: rgba(52, 152, 219, 0.1);
}

/* =====
   MENSAGENS DE FEEDBACK
===== */
.login-message {
  border-radius: var(--border-radius);
  padding: var(--spacing-md);
  margin-bottom: var(--spacing-lg);
  font-size: 0.9rem;
  text-align: center;
  animation: slideIn 0.3s ease;
}

.login-message.error {
  background: rgba(231, 76, 60, 0.15);
  color: #fc8181;
  border: 1px solid rgba(231, 76, 60, 0.3);
}

```

```

.login-message.success {
  background: rgba(46, 204, 113, 0.15);
  color: #68d391;
  border: 1px solid rgba(46, 204, 113, 0.3);
}

/* =====
   MODAL
===== */
.modal {
  display: none;
  position: fixed;
  z-index: 10000;
  left: 0;
  top: 0;
  width: 100%;
  height: 100%;
  background: rgba(0, 0, 0, 0.7);
  backdrop-filter: blur(4px);
  animation: fadeIn 0.3s ease;
}

.modal-content {
  position: relative;
  background: #ffffff;
  margin: 8% auto;
  border-radius: var(--border-radius-large);
  width: 90%;
  max-width: 480px;
  box-shadow: 0 25px 60px rgba(0, 0, 0, 0.4);
  animation: slideIn 0.3s ease-out;
  overflow: hidden;
  color: #2c3e50;
}

.modal-header {
  display: flex;

```

```

    justify-content: space-between;
    align-items: center;
    padding: var(--spacing-lg) var(--spacing-xl);
    background: var(--primary-color);
    color: var(--white);
}

.modal-header h3 {
    margin: 0;
    font-size: 1.2rem;
    font-weight: 600;
    color: #ffffff;
}

.modal-close {
    background: none;
    border: none;
    color: var(--white);
    font-size: 1.5rem;
    cursor: pointer;
    line-height: 1;
    padding: 0;
    opacity: 0.9;
    transition: opacity var(--transition-fast);
}

.modal-close:hover {
    opacity: 1;
}

.modal-form {
    padding: var(--spacing-xl);
}

.modal-form label {
    color: #2c3e50;
    font-size: 0.95rem;
    font-weight: 500;
}

```

```

    margin-bottom: 8px;
}

.modal-form input {
    background: #ffffff;
    border: 1px solid #cbd5e0;
    color: #2c3e50;
    border-radius: var(--border-radius);
    padding: 12px 16px;
    font-size: 1rem;
}

.modal-form input:focus {
    border-color: var(--primary-color);
    box-shadow: 0 0 0 3px rgba(52, 152, 219, 0.15);
}

.modal-buttons {
    display: flex;
    gap: var(--spacing-md);
    justify-content: flex-end;
    margin-top: var(--spacing-xl);
}

.btn-cancelar {
    background: #f7fafc;
    color: #4a5568;
    border: 1px solid #e2e8f0;
    padding: 10px 20px;
    border-radius: var(--border-radius);
    cursor: pointer;
    font-size: 0.95rem;
    font-weight: 500;
    transition: all var(--transition-fast);
}

.btn-cancelar:hover {
    background: #edf2f7;
}

```

```

}

.btn-enviar {
  background: var(--primary-color);
  color: var(--white);
  border: none;
  padding: 10px 24px;
  border-radius: var(--border-radius);
  cursor: pointer;
  font-size: 0.95rem;
  font-weight: 600;
  transition: all var(--transition-fast);
}

.btn-enviar:hover:not(:disabled) {
  background: var(--primary-dark);
}

.btn-enviar:disabled {
  opacity: 0.6;
  cursor: not-allowed;
}

/* =====
   TELA DE SUCESSO DO RESET
===== */
.reset-success {
  padding: var(--spacing-xl);
  text-align: center;
  color: #2c3e50;
}

.success-icon {
  font-size: 3rem;
  margin-bottom: var(--spacing-lg);
}

.reset-success h4 {

```

```

    color: var(--success-color);
    margin-bottom: var(--spacing-md);
    font-size: 1.25rem;
}

.protocolo-info {
    background: rgba(46, 204, 113, 0.1);
    padding: var(--spacing-lg);
    border-radius: var(--border-radius);
    margin-bottom: var(--spacing-lg);
    border-left: 4px solid var(--success-color);
}

.protocolo-info strong {
    font-size: 1.1rem;
    color: #27ae60;
    display: block;
    margin-bottom: var(--spacing-sm);
}

.btn-fechar {
    background: var(--success-color);
    color: var(--white);
    border: none;
    padding: 10px 24px;
    border-radius: var(--border-radius);
    cursor: pointer;
    font-weight: 500;
    transition: all var(--transition-fast);
}

.btn-fechar:hover {
    background: #27ae60;
}

/* =====
   RESPONSIVO
   ===== */

```

```

@media (max-width: 768px) {
  .login-box {
    padding: var(--spacing-xl);
    max-width: 100%;
  }
  .modal-content {
    width: 95%;
    margin: 15% auto;
  }
  .modal-buttons {
    flex-direction: column;
  }
  .modal-buttons button {
    width: 100%;
  }
}

@media (max-width: 480px) {
  .login-container {
    padding: var(--spacing-md);
    align-items: flex-start;
    padding-top: 40px;
  }
  .login-box {
    padding: var(--spacing-lg);
  }
  .login-header h1 {
    font-size: 1.5rem;
  }
  .btn-login {
    padding: 14px;
  }
}

```

6.3 — public/assets/css/dashboard.css

Por quê? Estilos exclusivos do dashboard: header com fundo #34495e, cards de estatísticas com número grande em azul #3498db, grid de acesso rápido.

Responsivo com breakpoints em 1024px, 768px e 480px com colapso progressivo das colunas.

```
/* =====  
  HEADER  
===== */  
.header {  
  background-color: #34495e;  
  padding: 15px 0;  
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);  
}  
  
.header-content {  
  max-width: 1200px;  
  margin: 0 auto;  
  padding: 0 20px;  
  display: flex;  
  justify-content: space-between;  
  align-items: center;  
}  
  
.header h1 {  
  color: #ecf0f1;  
  font-size: 22px;  
  margin: 0;  
}  
  
.user-info {  
  display: flex;  
  align-items: center;  
  gap: 20px;  
  color: #ecf0f1;  
  font-size: 14px;  
}  
  
.btn-sair {  
  padding: 8px 20px;  
  background-color: #e74c3c;
```

```

    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    font-size: 14px;
    transition: background-color 0.3s;
}

.btn-sair:hover {
    background-color: #c0392b;
}

/* =====
   CONTEÚDO PRINCIPAL
===== */
.main-content {
    max-width: 1200px;
    margin: 30px auto;
    padding: 0 20px;
}

.main-content h2 {
    color: #ecf0f1;
    font-size: 26px;
    margin-bottom: 25px;
}

/* =====
   CARDS DE ESTATÍSTICAS
===== */
.stats-container {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(220px, 1fr));
    gap: 20px;
    margin-bottom: 40px;
}

```

```

.stat-card {
  background-color: #34495e;
  padding: 30px 25px;
  border-radius: 12px;
  text-align: center;
  box-shadow: 0 4px 15px rgba(0, 0, 0, 0.2);
}

.stat-icon {
  font-size: 50px;
  margin-bottom: 12px;
}

.stat-card h3 {
  font-size: 42px;
  color: #3498db;
  margin: 12px 0;
  font-weight: 700;
}

.stat-card p {
  font-size: 16px;
  color: #bdc3c7;
  margin: 0;
  font-weight: 500;
}

/* =====
   ACESO RÁPIDO
===== */
.quick-access {
  margin-top: 40px;
}

.quick-access h3 {
  color: #ecf0f1;
  font-size: 22px;
  margin-bottom: 20px;
}

```

```

}

.quick-buttons {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));
  gap: 18px;
}

.quick-btn {
  background-color: #34495e;
  padding: 25px 18px;
  border-radius: 10px;
  text-align: center;
  text-decoration: none;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
}

.quick-btn span {
  font-size: 36px;
  display: block;
  margin-bottom: 10px;
}

.quick-btn p {
  color: #ecf0f1;
  font-size: 15px;
  margin: 0;
  font-weight: 500;
}

/* =====
   RESPONSIVO
===== */
@media (max-width: 1024px) {
  .header h1 { font-size: 20px; }
  .stat-card h3 { font-size: 38px; }
  .stats-container { grid-template-columns: repeat

```

```

(auto-fit, minmax(200px, 1fr)); }
}

@media (max-width: 768px) {
  .header-content { flex-direction: column; gap: 1
5px; text-align: center; }
  .user-info { flex-direction: column; gap: 1
0px; }
  .main-content h2 { font-size: 22px; }
  .stats-container { grid-template-columns: 1fr; }
  .quick-buttons { grid-template-columns: repeat
(2, 1fr); }
}

@media (max-width: 480px) {
  .main-content h2 { font-size: 20px; }
  .stat-card h3 { font-size: 32px; }
  .quick-buttons { grid-template-columns: 1fr; }
}

```

FASE 7 — Frontend JavaScript

7.1 — `public/assets/js/config.js`

Por quê primeiro entre os JS? Base para todos os outros scripts. Define `API_BASE_URL` detectando automaticamente se o acesso é via Node porta 3000 ou Live Server porta 5500, evitando erro de CORS. Exporta `apiRequest`, `isValidEmail` e `isValidPassword` como funções globais via `window`.

```

// public/assets/js/config.js

/* =====
  DETECÇÃO DE AMBIENTE
===== */
// Se porta 3000 = Node.js | Se porta 5500 = Live Server
const API_BASE_URL =
  window.location.port === '3000'

```

```

? `${window.location.origin}/api`
: 'http://127.0.0.1:3000/api';

/* =====
CONFIGURAÇÕES GLOBAIS
===== */
const CONFIG = {
  api: {
    baseUrl: API_BASE_URL,
    timeout: 30000,
    headers: { 'Content-Type': 'application/json' },
  },
  auth: {
    sessionKey: 'usuario',
    tokenExpiry: 24 * 60 * 60 * 1000, // 24 horas em ms
  },
  validation: {
    minPasswordLength: 6,
    emailRegex: /^[^\s@]+@[^\s@]+\.[^\s@]+$/,
  },
  messages: {
    networkError: 'Erro de conexão. Verifique sua internet.',
    serverError: 'Erro interno do servidor. Tente novamente.',
    sessionExpired: 'Sessão expirada. Faça login novamente.',
    invalidCredentials: 'Email ou senha inválidos.',
  },
};

/* =====
HELPER DE REQUISIÇÕES HTTP
===== */
const apiRequest = async (endpoint, options = {}) => {
  const url = `${CONFIG.api.baseUrl}${endpoint}`;

  const defaultOptions = {

```

```

    method: 'GET',
    headers: { ...CONFIG.api.headers },
  };

  const finalOptions = { ...defaultOptions, ...options };

  // Serializa body automaticamente
  if (finalOptions.body && typeof finalOptions.body === 'object') {
    finalOptions.body = JSON.stringify(finalOptions.body);
  }

  try {
    const response = await fetch(url, finalOptions);

    if (!response.ok) {
      const errorData = await response.json().catch(() => ({}));
      throw new Error(errorData.erro || `HTTP ${response.status}`);
    }

    return await response.json();
  } catch (error) {
    if (error.name === 'TypeError') {
      throw new Error(CONFIG.messages.networkError);
    }
    throw error;
  }
};

/* =====
   HELPERS DE VALIDAÇÃO
   ===== */
const isValidEmail = (email) =>
  CONFIG.validation.emailRegex.test(email);

```

```

const isValidPassword = (password) =>
  password && password.length >= CONFIG.validation.minPasswordLength;

/* =====
EXPORTAR GLOBALMENTE
===== */
window.CONFIG = CONFIG;
window.apiRequest = apiRequest;
window.isValidEmail = isValidEmail;
window.isValidPassword = isValidPassword;

```

7.2 — public/assets/js/auth.js

Por quê? Proteção de rotas do frontend. Verifica sessão válida no `sessionStorage` ao carregar páginas protegidas. Redireciona para login com URL absoluta se sessão estiver ausente, corrompida ou expirada (24h). Verificação periódica a cada 5 minutos e detecção de logout em outras abas via evento `storage`. O botão `btnSair` é gerenciado exclusivamente aqui — não no `dashboard.js` — para evitar duplo `confirm`.

```

// public/assets/js/auth.js
document.addEventListener('DOMContentLoaded', () => {

  /* =====
  CONFIGURAÇÃO
  ===== */
  const AUTH_CONFIG = {
    sessionKey: 'usuario',
    maxSessionTime: 24 * 60 * 60 * 1000, // 24 horas
    checkInterval: 5 * 60 * 1000, // verifica a cada 5 min
  };

  /* =====
  RECUPERAR SESSÃO
  ===== */
  const getUsuarioLogado = () => {

```

```

    try {
        const usuarioJson = sessionStorage.getItem(AUTH_CONFIG.sessionKey);
        if (!usuarioJson) return null;

        const sessionData = JSON.parse(usuarioJson);

        // Valida estrutura da sessão
        if (!sessionData.usuario || !sessionData.loginTime) {
            clearSession();
            return null;
        }

        // Valida expiração (24h)
        if (Date.now() - sessionData.loginTime > AUTH_CONFIG.maxSessionTime) {
            clearSession();
            return null;
        }

        return sessionData.usuario;
    } catch (error) {
        clearSession();
        return null;
    }
};

/* =====
LIMPAR SESSÃO
===== */
const clearSession = () => {
    sessionStorage.removeItem(AUTH_CONFIG.sessionKey);
    localStorage.removeItem('usuarioLogado');
};

/* =====
EXIBIR MENSAGEM
===== */

```

```

===== */
const showMessage = (message, type = 'error') => {
  const existingAlert = document.querySelector('.auth-alert');
  if (existingAlert) existingAlert.remove();

  const alert = document.createElement('div');
  alert.className = `auth-alert alert-${type}`;
  alert.style.cssText = `
    position: fixed;
    top: 20px;
    right: 20px;
    padding: 15px;
    background: ${type === 'error' ? '#e74c3c' : '#2ecc71'};
    color: white;
    border-radius: 5px;
    box-shadow: 0 2px 10px rgba(0,0,0,0.3);
    z-index: 9999;
    max-width: 300px;
  `;
  alert.textContent = message;
  document.body.appendChild(alert);
  setTimeout(() => alert.remove(), 4000);
};

/* =====
   REDIRECIONAR PARA LOGIN
===== */
const redirectToLogin = () => {
  if (!window.location.pathname.includes('login.html')) {
    window.location.href = 'http://127.0.0.1:3000/pages/login.html';
  }
};

/* =====
   VERIFICAR AUTENTICAÇÃO
===== */

```

```

===== */
const verificarAutenticacao = () => {
  const usuario = getUsuarioLogado();

  if (!usuario) {
    redirectToLogin();
    return false;
  }

  // Exibe nome no header
  const nomeEl = document.getElementById('nomeUsuario');
  if (nomeEl) {
    nomeEl.textContent = usuario.NomeCompleto || usuario.
Login || 'Usuário';
  }

  return true;
};

/* =====
LOGOUT
===== */
const logout = () => {
  try {
    clearSession();
    showMessage('Logout realizado com sucesso.', 'succes
s');
    setTimeout(() => {
      window.location.href = 'http://127.0.0.1:3000/page
s/login.html';
    }, 1500);
  } catch (error) {
    clearSession();
    window.location.href = 'http://127.0.0.1:3000/pages/l
ogin.html';
  }
};

```

```

/* =====
    INICIALIZAÇÃO
===== */
const currentPath = window.location.pathname;
const isProtectedPage =
    currentPath.includes('dashboard.html') ||
    currentPath.includes('admin.html');

if (isProtectedPage) {
    const isAuthenticated = verificarAutenticacao();

    // Verificação periódica a cada 5 minutos
    if (isAuthenticated) {
        setInterval(() => {
            if (!getUsuarioLogado()) {
                showMessage('Sessão perdida. Redirecionando...');
                setTimeout(redirectToLogin, 2000);
            }
        }, AUTH_CONFIG.checkInterval);
    }
}

/* =====
    BOTÃO SAIR
    (gerenciado aqui – não no dashboard.js)
===== */
const btnSair = document.getElementById('btnSair');
if (btnSair) {
    btnSair.addEventListener('click', (e) => {
        e.preventDefault();
        if (confirm('Deseja realmente sair do sistema?')) {
            logout();
        }
    });
}

/* =====
    DETECÇÃO DE LOGOUT EM OUTRA ABA
===== */

```

```

===== */
window.addEventListener('storage', (e) => {
    if (e.key === AUTH_CONFIG.sessionKey && !e.newValue &&
isProtectedPage) {
        showMessage('Sessão encerrada em outra aba.');
```

```

        setTimeout(redirectToLogin, 2000);
    }
});

/* =====
EXPORTAR FUNÇÕES GLOBALMENTE
===== */
window.logout = logout;
window.getUsuarioLogado = getUsuarioLogado;
window.verificarAutenticacao = verificarAutenticacao;

});

```

7.3 — public/assets/js/login.js

Por quê? Lógica completa do formulário de login. Bloqueia acesso se já houver sessão ativa. Gerencia estados de loading. Trata erros HTTP com mensagens contextualizadas (401, 429, erro de rede). Salva sessão no `sessionStorage` com timestamp e redireciona via URL absoluta para evitar conflito com Live Server.

```

// public/assets/js/login.js
document.addEventListener('DOMContentLoaded', () => {

    /* =====
    REDIRECIONAR SE JÁ LOGADO
    ===== */
    if (sessionStorage.getItem('usuario')) {
        window.location.href = 'http://127.0.0.1:3000/pages/das
hboard.html';
        return;
    }
}

```

```

/* =====
ELEMENTOS DO DOM
===== */
const form      = document.getElementById('formLogin');
const inputEmail = document.getElementById('email');
const inputSenha = document.getElementById('senha');
const toggleSenha = document.getElementById('toggleSenha');
const btnLogin   = form.querySelector('button[type="submit"]');

/* =====
TOGGLE DE SENHA
===== */
inputSenha.type = 'password';
toggleSenha.innerHTML = `
  <svg width="22" height="22" viewBox="0 0 24 24" fill="none" xmlns="http://www.w3.org/2000/svg">
    <path d="M2 12s3.5-7 10-7 10 7 10 7-3.5 7-10 7S2 12 2 12"
      stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round"/>
    <path d="M12 15a3 3 0 1 0 0-6 3 3 0 0 0 0 6"
      stroke="currentColor" stroke-width="2" stroke-linecap="round" stroke-linejoin="round"/>
    <path d="M4 4L20 20" stroke="currentColor" stroke-width="2" stroke-linecap="round"/>
  </svg>
`;

toggleSenha.addEventListener('click', () => {
  inputSenha.type = inputSenha.type === 'password' ? 'text' : 'password';
});

/* =====
EXIBIR MENSAGEM
===== */

```

```

const showMessage = (message, type = 'error') => {
  const existing = document.querySelector('.login-message');
  if (existing) existing.remove();

  const el = document.createElement('div');
  el.className = `login-message ${type}`;
  el.textContent = message;
  form.insertBefore(el, btnLogin);

  setTimeout(() => el.remove(), 5000);
};

/* =====
   ESTADO DE LOADING
===== */
const setLoadingState = (loading) => {
  btnLogin.disabled = loading;
  btnLogin.textContent = loading ? 'Entrando...' : 'Entrar';
  inputEmail.disabled = loading;
  inputSenha.disabled = loading;
};

/* =====
   SALVAR SESSÃO
===== */
const saveUserSession = (usuario) => {
  sessionStorage.setItem('usuario', JSON.stringify({
    usuario,
    loginTime: Date.now(),
    lastActivity: Date.now(),
  }));
};

/* =====
   SUBMIT DO FORMULÁRIO
===== */

```

```

form.addEventListener('submit', async (e) => {
  e.preventDefault();

  const email = inputEmail.value.trim();
  const senha = inputSenha.value;

  if (!email || !senha) {
    showMessage('Usuário e senha são obrigatórios', 'erro
r');
    return;
  }

  setLoadingState(true);

  try {
    const usuario = await apiRequest('/usuarios/login', {
      method: 'POST',
      body: { email, senha },
    });

    showMessage('Login realizado com sucesso!', 'succes
s');
    saveUserSession(usuario);

    setTimeout(() => {
      window.location.href = 'http://127.0.0.1:3000/page
s/dashboard.html';
    }, 1000);

  } catch (error) {
    if (error.message.includes('401') || error.message.in
cludes('inválidos')) {
      showMessage('Usuário ou senha incorretos', 'erro
r');
    } else if (error.message.includes('429')) {
      showMessage('Muitas tentativas. Aguarde alguns minu
tos.', 'error');
    } else {

```

```

        showMessage('Erro interno. Tente novamente.', 'erro
r');
    }
    } finally {
        setLoadingState(false);
    }
});

/* =====
   FOCO AUTOMÁTICO
   ===== */
setTimeout(() => inputEmail.focus(), 300);

});

```

7.4 — public/assets/js/dashboard.js

Por quê? Carrega os dados dinâmicos do dashboard após autenticação confirmada pelo `auth.js`. Exibe nome do usuário no header e busca total de usuários ativos via API. O `btnSair` não é gerenciado aqui — está exclusivamente no `auth.js` — para evitar o bug do duplo `confirm`.

```

// public/assets/js/dashboard.js
document.addEventListener('DOMContentLoaded', async () => {

    /* =====
       EXIBIR NOME DO USUÁRIO
       ===== */
    const sessionData = sessionStorage.getItem('usuario');

    if (sessionData) {
        const { usuario } = JSON.parse(sessionData);
        document.getElementById('nomeUsuario').textContent =
            usuario.NomeCompleto || usuario.Login || 'Usuário';
    }

    /* =====
       CARREGAR ESTATÍSTICAS

```

```

===== */
try {
    const usuarios = await apiRequest('/usuarios');
    document.getElementById('totalUsuarios').textContent =
usuarios.length;

} catch (error) {
    document.getElementById('totalUsuarios').textContent =
'0';
    console.error('Erro ao carregar usuários:', error);
}

});

```

FASE 8 — Checklist de Testes

Banco de Dados — SQL Server Management Studio

- ☐ Executar `CreateSoftwareProduct.sql` — banco `SoftwareProduct` criado sem erros
- ☐ Executar `Usuarios.sql` — tabela `dbo.Usuarios` criada sem erros
- ☐ Executar `TesteUsuarios.sql` — ciclo completo sem erros
- ☐ `SELECT * FROM dbo.Usuarios` retorna registros
- ☐ Campo `Senha` armazenado como hash bcrypt (não texto puro)
- ☐ Campo `Ativo = 1` nos registros ativos
- ☐ Executar UPDATE e verificar se `DataAtualizacao` mudou automaticamente (trigger)
- ☐ Índice `IX_Usuarios_Login` criado com sucesso

Backend — Postman

Pré-requisito: `npm install` executado e `npm run dev` rodando sem erros no terminal

- ☐ `POST http://127.0.0.1:3000/api/usuarios/login`
Body: `{ "email": "admin", "senha": "Senha@123" }`

Esperado: 200 OK — objeto do usuário **sem** campo Senha

☐ POST /api/usuarios/login com senha errada

Esperado: 401 Unauthorized

☐ POST /api/usuarios/login — 6ª tentativa seguida

Esperado: 429 Too Many Requests

☐ GET http://127.0.0.1:3000/api/usuarios

Esperado: 200 OK — array de usuários

☐ POST http://127.0.0.1:3000/api/usuarios

Body: { "login": "novo", "senha": "123456", "nomeCompleto": "Novo User", "email": "novo@teste.com" }

Esperado: 201 Created

☐ POST /api/usuarios com login duplicado

Esperado: 409 Conflict

☐ PUT http://127.0.0.1:3000/api/usuarios/1

Body: { "nomeCompleto": "Nome Atualizado", "email": "atualizado@teste.com", "senha": "123456" }

Esperado: 200 OK

☐ DELETE http://127.0.0.1:3000/api/usuarios/1

Esperado: 200 OK — verificar no banco que Ativo = 0

☐ POST /api/usuarios/reset-senha

Body: { "email": "teste01@email.com" }

Esperado: 200 OK com campo protocolo

☐ POST /api/qualquer sem header Content-Type: application/json

Esperado: 415 Unsupported Media Type

Frontend — Navegador

URL base: http://127.0.0.1:3000

☐ http://127.0.0.1:3000 redireciona para a tela de login

☐ Tela de login carrega sem erros no console (F12)

☐ Toggle de senha (ícone do olho) funciona

☐ Login com credenciais inválidas exibe mensagem de erro

☐ Login com credenciais válidas redireciona para o dashboard

- ☐ Dashboard exibe nome do usuário logado no header
 - ☐ Card "CLIENTES" exibe total correto de usuários
 - ☐ Botão "Sair" exibe `confirm` — apenas uma vez (sem duplo confirm)
 - ☐ Após logout, tentativa de acessar dashboard redireciona para login
 - ☐ Acesso direto a `/pages/dashboard.html` sem sessão redireciona para login
 - ☐ Modal "Esqueci minha senha" abre e fecha corretamente
 - ☐ Reset de senha exibe protocolo gerado
 - ☐ Layout responsivo em tela menor que 768px
-

FASE 9 — Versionamento GitHub

Comandos Utilizados em Todos os ACs

Estes são os comandos Git utilizados em todas as entregas do AC0 ao AC4. Aprenda uma vez, use sempre.

```
# Verificar estado atual dos arquivos
# Mostra quais arquivos foram modificados, adicionados ou e
# stão prontos para commit
# SEMPRE rodar antes de qualquer coisa
git status
```

```
# Adicionar todos os arquivos ao stage
# O ponto (.) significa "tudo na pasta atual"
# Para arquivo específico: git add server.js
git add .
```

```
# Verificar novamente — SEGURANÇA
# Confirmar que .env e Token GitHub.txt NÃO aparecem
# Se aparecerem, remover com os comandos abaixo
git status
```

```
# Remover arquivo do stage sem apagar o arquivo do disco
git restore --staged .env
```

```
git restore --staged "acs/Token GitHub.txt"
```

```
# Criar o commit com mensagem descritiva  
# Salva o snapshot do projeto com uma mensagem explicando o  
# que foi feito  
git commit -m "AC0: estrutura inicial completa"
```

```
# Enviar para o GitHub  
# Sobe os commits locais para o repositório remoto  
git push origin main
```

```
# Ver histórico de commits de forma resumida  
# Útil para revisar o que foi feito  
git log --oneline
```

```
# Desfazer último commit sem apagar os arquivos  
# Útil se errar a mensagem do commit  
git reset --soft HEAD~1
```

Passo a Passo — AC0

```
# 1. Entrar na pasta do projeto  
cd C:\software-product  
  
# 2. Verificar status  
git status  
  
# 3. Adicionar tudo  
git add .  
  
# 4. Verificar se .env não está no stage  
git status  
  
# 5. Commit  
git commit -m "AC0: estrutura inicial completa - backend +
```

```
frontend + banco"
```

```
# 6. Push
```

```
git push origin main
```

```
# 7. Confirmar no GitHub
```

```
# Acessar https://github.com/busellirogerio/software-product
```

```
# Verificar se os arquivos foram atualizados
```

Convenção de Mensagens de Commit

Prefixo	Quando usar
AC0:	Entrega acadêmica AC0
AC1:	Entrega acadêmica AC1
AC2:	Entrega acadêmica AC2
feat:	Nova funcionalidade
fix:	Correção de bug
refactor:	Refatoração sem mudança de comportamento
style:	Ajuste de CSS ou formatação
docs:	Atualização de documentação

FASE 10 — README

```
# 🗄 Troca0leo – Sistema de Gerenciamento de Oficina
```

Sistema acadêmico desenvolvido para a disciplina de Software Product na Faculdade Impacta – Análise e Desenvolvimento de Sistemas.

```
## Stack
```

- **Backend:** Node.js + Express.js
- **Banco de Dados:** SQL Server 2019+

- ****Frontend:**** HTML5 + CSS3 + JavaScript Vanilla
- ****Segurança:**** bcrypt + rate limiting + CORS + headers HTTP

Pré-requisitos

- Node.js 18+
- SQL Server 2019+
- SQL Server Management Studio

Instalação

1. Clone o repositório:

```
git clone https://github.com/busellirogerio/software-product.git
cd software-product
```

2. Instale as dependências:

```
npm install
```

3. Configure o banco de dados no SQL Server Management Studio:

- Execute: sql/CreateSoftwareProduct.sql
- Execute: sql/Usuarios.sql
- Execute: sql/TesteUsuarios.sql

4. Crie o arquivo .env na raiz do projeto:

```
PORT=3000
DB_SERVER=127.0.0.1
DB_DATABASE=SoftwareProduct
DB_USER=sa
DB_PASSWORD=sua_senha
DB_PORT=1433
```

5. Inicie o servidor:

```
npm run dev
```

6. Acesse no navegador:

http://127.0.0.1:3000

Estrutura de Pastas

software-product/

— sql/	Scripts SQL
— src/	
— config/	Pool de conexão
— repositories/	Queries SQL
— controllers/	Lógica HTTP
— routes/	Rotas da API
— public/	
— pages/	HTML
— assets/	
— css/	Estilos
— js/	Scripts
— .env	Credenciais (não versionado)
— server.js	Entrada da aplicação

Endpoints da API

Método	Endpoint	Descrição
-----	-----	-----
POST	/api/usuarios/login	Autenticação
POST	/api/usuarios/reset-senha	Reset de senha
GET	/api/usuarios	Listar usuários
POST	/api/usuarios	Criar usuário
PUT	/api/usuarios/:id	Atualizar usuário
DELETE	/api/usuarios/:id	Soft delete

Autor

Buselli Rogerio – Faculdade Impacta – ADS 2026

AC0 — Versão 2.0 / 18/02/2026 / 21 arquivos verificados e documentados