AC1 - Cadastro Clientes

AC1 - Cadastro de Clientes

AC1 — Cadastro de Clientes

Projeto: TrocaOleo — Sistema de Gerenciamento de Oficina
Instituição: Faculdade Impacta | **Curso:** ADS
Data: 19/02/2026 | **Versão:** 1.0
Autor: Buselli Rogerio

🎯 Objetivo do AC1

Implementar o módulo completo de gerenciamento de clientes, integrado ao dashboard existente como seção SPA (Single Page Application), garantindo:

- Cadastro completo de pessoas físicas e jurídicas
- Validação de CPF/CNPJ com formatação automática
- Operações CRUD com soft delete
- Busca por nome, CPF/CNPJ e telefone
- Reativação inteligente de clientes excluídos
- Interface accordion colapsável sem abertura de novas abas
- Listagem ordenável por nome

📦 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 AC1 — Resumo

Fase	O que é	Status
Fase 1	Bug Fix — CPF duplicado após soft delete	✅
Fase 2	Banco de Dados — tabela Clientes	✅
Fase 3	Backend — repository + controller + routes	✅
Fase 4	Atualização — server.js	✅
Fase 5	Frontend — dashboard.html (seção clientes)	✅
Fase 6	Frontend — clientes.css	✅
Fase 7	Frontend — clientes.js	✅
Fase 8	Checklist de Testes	✅
Fase 9	Versionamento GitHub	⏳

🚀 Mapa de Implantação — Ordem Prioritária

FASE 1 — Bug Fix: CPF Duplicado Após Soft Delete

> **Por quê primeiro?** Ao deletar um cliente (soft delete, `Ativo = 0`) e tentar recadastrá-lo com o mesmo CPF/CNPJ, o banco retornava erro de violação de UNIQUE — pois a constraint não distingue ativos de inativos. A solução adotada foi a reativação inteligente: antes de qualquer INSERT, o sistema verifica se já existe um registro inativo com aquele CPF/CNPJ; se existir, faz UPDATE reativando e atualizando os dados em vez de criar um novo registro.
>
> **Causa técnica adicional:** O `UPDATE` com cláusula `OUTPUT` direto em tabelas com trigger ativo é bloqueado pelo SQL Server (erro 334). A solução foi separar a operação em dois passos: UPDATE sem OUTPUT, seguido de SELECT para retornar o registro atualizado.

Arquivo alterado: `src/repositories/clienteRepository.js` — método `criar()`

Lógica implementada:

```
1. Recebe dados do novo cadastro
2. Busca CpfCnpj na tabela com Ativo = 0
3. SE encontrar → UPDATE reativando + SELECT para retornar
4. SE não encontrar → INSERT normal com OUTPUT
```

FASE 2 — Banco de Dados

2.1 — `sql/Clientes.sql`

> **Por quê?** Define a estrutura completa da tabela `dbo.Clientes` com suporte a pessoa física (CPF, 11 dígitos) e jurídica (CNPJ, 14 dígitos), campos de endereço completo, controle lógico via soft delete (`Ativo`) e auditoria automática via trigger. Todos os campos de texto são armazenados em maiúsculo para padronização. O CPF/CNPJ é armazenado sem formatação (apenas números) e tem constraint UNIQUE para evitar duplicatas ativas. Os 4 índices otimizam as buscas mais frequentes: por nome, CPF/CNPJ, telefone e status ativo.

```
-- ========================================
-- BANCO: SoftwareProduct
-- TABELA: dbo.Clientes
-- VERSÃO: 1.0 - AC1
-- DATA: 2026-02-19
-- ========================================

USE SoftwareProduct;
GO

-- ========================================
-- LIMPEZA — remove objetos existentes
-- ========================================
IF OBJECT_ID('dbo.TR_Clientes_SetDataAtualizacao', 'TR') IS
NOT NULL
    DROP TRIGGER dbo.TR_Clientes_SetDataAtualizacao;
GO

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

-- ========================================
-- CRIAÇÃO DA TABELA
-- ========================================
```

```
CREATE TABLE dbo.Clientes
(
    -- IDENTIFICAÇÃO
    ClienteId           INT IDENTITY(1,1)  NOT NULL,

    -- TIPO: PF = Pessoa Física | PJ = Pessoa Jurídica
    Tipo                CHAR(2)            NOT NULL
        CONSTRAINT CK_Clientes_Tipo CHECK (Tipo IN ('PF',
'PJ')),

    -- CPF (11 dígitos) ou CNPJ (14 dígitos) — sem formataç
ão, maiúsculo
    CpfCnpj             NVARCHAR(14)       NOT NULL,

    -- DADOS PESSOAIS — armazenados em maiúsculo
    NomeCompleto        NVARCHAR(120)      NOT NULL,
    DataNascimento      DATE               NULL,
    Genero              CHAR(1)            NULL
        CONSTRAINT CK_Clientes_Genero CHECK (Genero IS NULL
OR Genero IN ('M', 'F', 'O')),

    -- CONTATO
    Telefone            NVARCHAR(20)       NULL,
    TelefoneWhatsApp    BIT                NOT NULL
        CONSTRAINT DF_Clientes_TelefoneWhatsApp DEFAULT
(0),
    Email               NVARCHAR(254)      NULL
        CONSTRAINT CK_Clientes_Email CHECK (Email IS NULL O
R Email LIKE '%_@_%._%'),

    -- ENDEREÇO — armazenados em maiúsculo
    Cep                 CHAR(8)            NULL,
    Logradouro          NVARCHAR(150)      NULL,
    Numero              NVARCHAR(10)       NULL,
    Complemento         NVARCHAR(60)       NULL,
    Bairro              NVARCHAR(80)       NULL,
    Cidade              NVARCHAR(80)       NULL,
    Estado              CHAR(2)            NULL,
```

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

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

    -- CONSTRAINTS
    CONSTRAINT PK_Clientes
        PRIMARY KEY CLUSTERED (ClienteId),
    CONSTRAINT UQ_Clientes_CpfCnpj
        UNIQUE (CpfCnpj)
);
GO

-- ========================================
-- ÍNDICES — otimizam buscas frequentes
-- ========================================
CREATE NONCLUSTERED INDEX IX_Clientes_NomeCompleto
    ON dbo.Clientes (NomeCompleto)
    WHERE Ativo = 1;
GO

CREATE NONCLUSTERED INDEX IX_Clientes_CpfCnpj
    ON dbo.Clientes (CpfCnpj)
    WHERE Ativo = 1;
GO

CREATE NONCLUSTERED INDEX IX_Clientes_Telefone
    ON dbo.Clientes (Telefone)
    WHERE Ativo = 1;
GO
```

```
CREATE NONCLUSTERED INDEX IX_Clientes_Ativo
    ON dbo.Clientes (Ativo)
    INCLUDE (ClienteId, NomeCompleto, CpfCnpj);
GO

-- ========================================
-- TRIGGER – atualiza DataAtualizacao automaticamente
-- ========================================
CREATE TRIGGER dbo.TR_Clientes_SetDataAtualizacao
ON dbo.Clientes
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;
    UPDATE c
    SET DataAtualizacao = SYSDATETIME()
    FROM dbo.Clientes c
    INNER JOIN inserted i ON i.ClienteId = c.ClienteId;
END;
GO

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

2.2 — `sql/TesteClientes.sql`

> **Por quê?** Valida a tabela antes de subir o backend. Executa o ciclo completo de 13 passos: inserção de PF e PJ, verificação do trigger de auditoria, soft delete, buscas por nome/CPF/telefone e listagem geral. Garante que estrutura, constraints e trigger estão funcionando corretamente.

```
USE SoftwareProduct;
GO

-- 1) ZERAR TABELA
DELETE FROM dbo.Clientes;
```

```
DBCC CHECKIDENT ('dbo.Clientes', RESEED, 0);
GO

-- 2) VERIFICAR ZEROU
SELECT * FROM dbo.Clientes;
GO

-- 3) INSERIR CLIENTE PF
INSERT INTO dbo.Clientes
  (Tipo, CpfCnpj, NomeCompleto, DataNascimento, Genero,
   Telefone, TelefoneWhatsApp, Email,
   Cep, Logradouro, Numero, Complemento, Bairro, Cidade, Es
tado, Ativo)
VALUES
  ('PF', '98765432100', 'JOAO DA SILVA', '1985-03-15', 'M',
   '11987654321', 1, 'JOAO@EMAIL.COM',
   '01310100', 'AVENIDA PAULISTA', '1000', 'APTO 52', 'BELA
VISTA', 'SAO PAULO', 'SP', 1);
GO

-- 4) INSERIR CLIENTE PJ
INSERT INTO dbo.Clientes
  (Tipo, CpfCnpj, NomeCompleto,
   Telefone, TelefoneWhatsApp, Email,
   Cep, Logradouro, Numero, Bairro, Cidade, Estado, Ativo)
VALUES
  ('PJ', '12345678000195', 'EMPRESA TESTE LTDA',
   '1133334444', 0, 'CONTATO@EMPRESA.COM',
   '01001000', 'PRACA DA SE', '10', 'SE', 'SAO PAULO', 'S
P', 1);
GO

-- 5) VERIFICAR INSERÇÕES
SELECT * FROM dbo.Clientes;
GO

-- 6) ATUALIZAR (verificar trigger DataAtualizacao)
UPDATE dbo.Clientes
```

```
SET NomeCompleto = 'JOAO DA SILVA ATUALIZADO'
WHERE CpfCnpj = '98765432100';
GO

-- 7) VERIFICAR TRIGGER – DataAtualizacao deve ter mudado
SELECT ClienteId, NomeCompleto, DataCriacao, DataAtualizaca
o
FROM dbo.Clientes
WHERE CpfCnpj = '98765432100';
GO

-- 8) SOFT DELETE
UPDATE dbo.Clientes
SET Ativo = 0
WHERE CpfCnpj = '98765432100';
GO

-- 9) VERIFICAR SOFT DELETE – registro existe mas Ativo = 0
SELECT ClienteId, NomeCompleto, Ativo FROM dbo.Clientes;
GO

-- 10) BUSCAR POR NOME
SELECT ClienteId, NomeCompleto, CpfCnpj, Telefone
FROM dbo.Clientes
WHERE NomeCompleto LIKE '%EMPRESA%' AND Ativo = 1;
GO

-- 11) BUSCAR POR CPF/CNPJ
SELECT ClienteId, NomeCompleto, CpfCnpj
FROM dbo.Clientes
WHERE CpfCnpj = '12345678000195' AND Ativo = 1;
GO

-- 12) BUSCAR POR TELEFONE
SELECT ClienteId, NomeCompleto, Telefone
FROM dbo.Clientes
WHERE Telefone LIKE '%3333%' AND Ativo = 1;
GO
```

```
-- 13) LISTAR TODOS ATIVOS
SELECT ClienteId, Tipo, CpfCnpj, NomeCompleto, Cidade, Esta
do
FROM dbo.Clientes
WHERE Ativo = 1
ORDER BY NomeCompleto;
GO
```

FASE 3 — Backend

3.1 — `src/repositories/clienteRepository.js`

> **Por quê?** Camada exclusiva de acesso ao banco para clientes. Toda query SQL fica centralizada aqui. Implementa os métodos de listagem, busca (por nome, CPF/CNPJ e telefone), criação com reativação inteligente, atualização e soft delete. A busca remove formatação automaticamente antes de consultar. O método `criar()` verifica existência de registro inativo antes de inserir, reativando se necessário — e por conta do trigger na tabela, o UPDATE é feito sem `OUTPUT`, seguido de `SELECT` separado para retornar o registro.

```
// src/repositories/clienteRepository.js

const { getPool, sql } = require('../config/database');

class ClienteRepository {

  /* ============================
     LISTAR TODOS
     Retorna apenas clientes ativos
     ordenados por nome
  ============================ */
  async listarTodos() {
    const pool = await getPool();
    const result = await pool.request().query(`
        SELECT
```

```
          ClienteId, Tipo, CpfCnpj, NomeCompleto, DataNasci
mento,
          Genero, Telefone, TelefoneWhatsApp, Email,
          Cep, Logradouro, Numero, Complemento, Bairro, Cid
ade, Estado, DataCriacao
        FROM dbo.Clientes
        WHERE Ativo = 1
        ORDER BY NomeCompleto
      `);
    return result.recordset;
  }

  /* ============================
     BUSCAR POR ID
  ============================ */
  async buscarPorId(id) {
    const pool = await getPool();
    const result = await pool
      .request()
      .input('id', sql.Int, id)
      .query(`
        SELECT *
        FROM dbo.Clientes
        WHERE ClienteId = @id AND Ativo = 1
      `);
    return result.recordset[0] || null;
  }

  /* ============================
     BUSCAR POR NOME
     Busca parcial — considera espaços
     Converte input para maiúsculo
  ============================ */
  async buscarPorNome(nome) {
    const pool = await getPool();
    const result = await pool
      .request()
      .input('nome', sql.NVarChar, '%' + nome.toUpperCase()
```

```
+ '%')
      .query(`
        SELECT ClienteId, Tipo, CpfCnpj, NomeCompleto, Gene
ro, Telefone, DataNascimento
        FROM dbo.Clientes
        WHERE NomeCompleto LIKE @nome AND Ativo = 1
        ORDER BY NomeCompleto
      `);
    return result.recordset;
  }

  /* ============================
     BUSCAR POR CPF/CNPJ
     Remove formatação antes de buscar
  ============================ */
  async buscarPorCpfCnpj(cpfCnpj) {
    const pool = await getPool();
    const apenasNumeros = cpfCnpj.replace(/[.\-\/]/g, '');
    const result = await pool
      .request()
      .input('cpfCnpj', sql.NVarChar, apenasNumeros)
      .query(`
        SELECT ClienteId, Tipo, CpfCnpj, NomeCompleto, Gene
ro, Telefone, DataNascimento
        FROM dbo.Clientes
        WHERE CpfCnpj = @cpfCnpj AND Ativo = 1
      `);
    return result.recordset;
  }

  /* ============================
     BUSCAR POR TELEFONE
     Busca parcial
  ============================ */
  async buscarPorTelefone(telefone) {
    const pool = await getPool();
    const apenasNumeros = telefone.replace(/[\s\-\(\)]/g,
'');
```

```
    const result = await pool
      .request()
      .input('telefone', sql.NVarChar, '%' + apenasNumeros
+ '%')
      .query(`
        SELECT ClienteId, Tipo, CpfCnpj, NomeCompleto, Gene
ro, Telefone, DataNascimento
        FROM dbo.Clientes
        WHERE Telefone LIKE @telefone AND Ativo = 1
        ORDER BY NomeCompleto
      `);
    return result.recordset;
  }

  /* ============================
     CRIAR
     Se CPF/CNPJ já existe com Ativo = 0
     reativa e atualiza os dados (2 passos)
     Se não existe — INSERT normal
     ATENÇÃO: UPDATE sem OUTPUT direto
     pois a tabela tem trigger ativo
  ============================ */
  async criar(dados) {
    const pool = await getPool();

    // Verifica se já existe inativo com esse CPF/CNPJ
    const existente = await pool
      .request()
      .input('cpfCnpj', sql.NVarChar, dados.cpfCnpj)
      .query(`
        SELECT ClienteId
        FROM dbo.Clientes
        WHERE CpfCnpj = @cpfCnpj AND Ativo = 0
      `);

    if (existente.recordset[0]) {
      // PASSO 1 — Reativa e atualiza dados (sem OUTPUT por
causa do trigger)
```

```
      const id = existente.recordset[0].ClienteId;
      await pool
        .request()
        .input('id',               sql.Int,      id)
        .input('nomeCompleto',     sql.NVarChar, dados.nome
Completo)
        .input('dataNascimento',   sql.Date,     dados.data
Nascimento   || null)
        .input('genero',           sql.Char,     dados.gene
ro           || null)
        .input('telefone',         sql.NVarChar, dados.tele
fone         || null)
        .input('telefoneWhatsApp', sql.Bit,      dados.tele
foneWhatsApp ? 1 : 0)
        .input('email',            sql.NVarChar, dados.emai
l            || null)
        .input('cep',              sql.Char,     dados.cep
|| null)
        .input('logradouro',       sql.NVarChar, dados.logr
adouro       || null)
        .input('numero',           sql.NVarChar, dados.nume
ro           || null)
        .input('complemento',      sql.NVarChar, dados.comp
lemento      || null)
        .input('bairro',           sql.NVarChar, dados.bair
ro           || null)
        .input('cidade',           sql.NVarChar, dados.cida
de           || null)
        .input('estado',           sql.Char,     dados.esta
do           || null)
        .query(`
          UPDATE dbo.Clientes
          SET
            Ativo           = 1,
            NomeCompleto    = @nomeCompleto,
            DataNascimento  = @dataNascimento,
            Genero          = @genero,
            Telefone        = @telefone,
```

```
            TelefoneWhatsApp = @telefoneWhatsApp,
            Email            = @email,
            Cep              = @cep,
            Logradouro       = @logradouro,
            Numero           = @numero,
            Complemento      = @complemento,
            Bairro           = @bairro,
            Cidade           = @cidade,
            Estado           = @estado
          WHERE ClienteId = @id
        `);

      // PASSO 2 — Busca o registro reativado para retornar
      const reativado = await pool
        .request()
        .input('id', sql.Int, id)
        .query(`SELECT * FROM dbo.Clientes WHERE ClienteId
= @id`);
      return reativado.recordset[0];
    }

    // INSERT NORMAL — CPF/CNPJ novo
    const result = await pool
      .request()
      .input('tipo',             sql.Char,     dados.tipo)
      .input('cpfCnpj',          sql.NVarChar, dados.cpfCnp
j)
      .input('nomeCompleto',     sql.NVarChar, dados.nomeCo
mpleto)
      .input('dataNascimento',   sql.Date,     dados.dataNa
scimento   || null)
      .input('genero',           sql.Char,     dados.genero
|| null)
      .input('telefone',         sql.NVarChar, dados.telefo
ne         || null)
      .input('telefoneWhatsApp', sql.Bit,      dados.telefo
neWhatsApp ? 1 : 0)
      .input('email',            sql.NVarChar, dados.email
```

```
|| null)
      .input('cep',              sql.Char,     dados.cep
|| null)
      .input('logradouro',       sql.NVarChar, dados.lograd
ouro       || null)
      .input('numero',           sql.NVarChar, dados.numero
|| null)
      .input('complemento',      sql.NVarChar, dados.comple
mento      || null)
      .input('bairro',           sql.NVarChar, dados.bairro
|| null)
      .input('cidade',           sql.NVarChar, dados.cidade
|| null)
      .input('estado',           sql.Char,     dados.estado
|| null)
      .query(`
        INSERT INTO dbo.Clientes
          (Tipo, CpfCnpj, NomeCompleto, DataNascimento, Gen
ero,
           Telefone, TelefoneWhatsApp, Email,
           Cep, Logradouro, Numero, Complemento, Bairro, Ci
dade, Estado, Ativo)
        OUTPUT
          INSERTED.ClienteId, INSERTED.Tipo, INSERTED.CpfCn
pj,
          INSERTED.NomeCompleto, INSERTED.DataNascimento, I
NSERTED.Genero,
          INSERTED.Telefone, INSERTED.TelefoneWhatsApp, INS
ERTED.Email,
          INSERTED.Cep, INSERTED.Logradouro, INSERTED.Numer
o,
          INSERTED.Complemento, INSERTED.Bairro, INSERTED.C
idade,
          INSERTED.Estado, INSERTED.Ativo, INSERTED.DataCri
acao
        VALUES
          (@tipo, @cpfCnpj, @nomeCompleto, @dataNascimento,
@genero,
```

```
            @telefone, @telefoneWhatsApp, @email,
            @cep, @logradouro, @numero, @complemento, @bairr
o, @cidade, @estado, 1)
        `);
      return result.recordset[0];
    }

    /* ============================
       ATUALIZAR
    ============================ */
    async atualizar(id, dados) {
      const pool = await getPool();
      const result = await pool
        .request()
        .input('id',              sql.Int,      id)
        .input('nomeCompleto',    sql.NVarChar, dados.nomeCo
mpleto)
        .input('dataNascimento',  sql.Date,     dados.dataNa
scimento   || null)
        .input('genero',          sql.Char,     dados.genero
|| null)
        .input('telefone',        sql.NVarChar, dados.telefo
ne         || null)
        .input('telefoneWhatsApp', sql.Bit,     dados.telefo
neWhatsApp ? 1 : 0)
        .input('email',           sql.NVarChar, dados.email
|| null)
        .input('cep',             sql.Char,     dados.cep
|| null)
        .input('logradouro',      sql.NVarChar, dados.lograd
ouro       || null)
        .input('numero',          sql.NVarChar, dados.numero
|| null)
        .input('complemento',     sql.NVarChar, dados.comple
mento      || null)
        .input('bairro',          sql.NVarChar, dados.bairro
|| null)
        .input('cidade',          sql.NVarChar, dados.cidade
```

```
|| null)
      .input('estado',           sql.Char,     dados.estado
|| null)
      .query(`
        UPDATE dbo.Clientes
        SET
          NomeCompleto     = @nomeCompleto,
          DataNascimento   = @dataNascimento,
          Genero           = @genero,
          Telefone         = @telefone,
          TelefoneWhatsApp = @telefoneWhatsApp,
          Email            = @email,
          Cep              = @cep,
          Logradouro       = @logradouro,
          Numero           = @numero,
          Complemento      = @complemento,
          Bairro           = @bairro,
          Cidade           = @cidade,
          Estado           = @estado
        WHERE ClienteId = @id AND Ativo = 1
      `);
    return result.rowsAffected[0];
  }

  /* ============================
     DELETAR (soft delete)
     Ativo = 0 — registro permanece
  ============================ */
  async deletar(id) {
    const pool = await getPool();
    const result = await pool
      .request()
      .input('id', sql.Int, id)
      .query(`
        UPDATE dbo.Clientes
        SET Ativo = 0
        WHERE ClienteId = @id
      `);
```

```
    return result.rowsAffected[0];
  }

}

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

3.2 — `src/controllers/clienteController.js`

> **Por quê?** Camada intermediária entre rotas e repository. Valida todos os campos recebidos, converte textos para maiúsculo antes de gravar, remove formatação de CPF/CNPJ (aceita com ou sem pontuação), valida comprimento do CPF (11) e CNPJ (14), formato de email e gênero. Retorna status HTTP semânticos: 400 (dados inválidos), 404 (não encontrado), 409 (CPF/CNPJ já ativo), 500 (erro interno). A busca unificada aceita o parâmetro `tipo` (nome, cpfcnpj, telefone) via query string.

```
// src/controllers/clienteController.js

const clienteRepository = require('../repositories/clienteR
epository');

class ClienteController {

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

```
  }

  /* ============================
     BUSCAR (unificada)
     Query: ?tipo=nome&valor=joao
     Query: ?tipo=cpfcnpj&valor=12345678901
     Query: ?tipo=telefone&valor=11999999999
  ============================ */
  async buscar(req, res) {
    try {
      const { tipo, valor } = req.query;

      if (!tipo || !valor) {
        return res.status(400).json({ erro: 'Parâmetros tip
o e valor são obrigatórios' });
      }

      let resultado;

      if (tipo === 'nome') {
        resultado = await clienteRepository.buscarPorNome(v
alor);
      } else if (tipo === 'cpfcnpj') {
        resultado = await clienteRepository.buscarPorCpfCnp
j(valor);
      } else if (tipo === 'telefone') {
        resultado = await clienteRepository.buscarPorTelefo
ne(valor);
      } else {
        return res.status(400).json({ erro: 'Tipo de busca
inválido. Use: nome, cpfcnpj ou telefone' });
      }

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

```
    }
  }

  /* ============================
     BUSCAR POR ID
  ============================ */
  async buscarPorId(req, res) {
    try {
      const { id } = req.params;

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

      const cliente = await clienteRepository.buscarPorId(p
arseInt(id));

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

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

  /* ============================
     CRIAR
     Valida campos obrigatórios
     Converte textos para maiúsculo
     Remove formatação do CPF/CNPJ
  ============================ */
```

```
  async criar(req, res) {
    try {
      const {
        tipo, cpfCnpj, nomeCompleto, dataNascimento, gener
o,
        telefone, telefoneWhatsApp, email,
        cep, logradouro, numero, complemento, bairro, cidad
e, estado
      } = req.body;

      // Validações obrigatórias
      if (!tipo || !cpfCnpj || !nomeCompleto) {
        return res.status(400).json({ erro: 'Tipo, CPF/CNPJ
e Nome são obrigatórios' });
      }

      if (!['PF', 'PJ'].includes(tipo.toUpperCase())) {
        return res.status(400).json({ erro: 'Tipo deve ser
PF ou PJ' });
      }

      // Remove formatação do CPF/CNPJ
      const cpfCnpjLimpo = cpfCnpj.replace(/[.\-\/]/g, '');

      // Valida comprimento
      if (tipo === 'PF' && cpfCnpjLimpo.length !== 11) {
        return res.status(400).json({ erro: 'CPF deve ter 1
1 dígitos' });
      }
      if (tipo === 'PJ' && cpfCnpjLimpo.length !== 14) {
        return res.status(400).json({ erro: 'CNPJ deve ter
14 dígitos' });
      }

      // Valida email se informado
      if (email) {
        const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
        if (!emailRegex.test(email)) {
```

```
          return res.status(400).json({ erro: 'Formato de e
mail inválido' });
        }
      }

      // Valida gênero se informado
      if (genero && !['M', 'F', 'O'].includes(genero.toUppe
rCase())) {
        return res.status(400).json({ erro: 'Gênero deve se
r M, F ou O' });
      }

      // Monta dados convertendo textos para maiúsculo
      const dados = {
        tipo:             tipo.toUpperCase(),
        cpfCnpj:          cpfCnpjLimpo,
        nomeCompleto:     nomeCompleto.toUpperCase().trim
(),
        dataNascimento:   dataNascimento || null,
        genero:           genero ? genero.toUpperCase() : n
ull,
        telefone:         telefone ? telefone.replace(/[\s
\-\(\)]/g, '') : null,
        telefoneWhatsApp: telefoneWhatsApp ? 1 : 0,
        email:            email ? email.toUpperCase().trim
() : null,
        cep:              cep ? cep.replace(/[.\-]/g, '') :
null,
        logradouro:       logradouro ? logradouro.toUpperCa
se().trim() : null,
        numero:           numero ? numero.toUpperCase().tri
m() : null,
        complemento:      complemento ? complemento.toUpper
Case().trim() : null,
        bairro:           bairro ? bairro.toUpperCase().tri
m() : null,
        cidade:           cidade ? cidade.toUpperCase().tri
m() : null,
```

```
        estado:           estado ? estado.toUpperCase().tri
m() : null,
      };

      const cliente = await clienteRepository.criar(dados);
      res.status(201).json(cliente);

    } catch (error) {
      if (error.message && error.message.includes('UNIQU
E')) {
        return res.status(409).json({ erro: 'CPF/CNPJ já ca
dastrado e ativo' });
      }
      console.error('Erro ao criar cliente:', error);
      res.status(500).json({ erro: 'Erro interno do servido
r' });
    }
  }

  /* ============================
     ATUALIZAR
     CPF/CNPJ e Tipo não podem ser alterados
  ============================ */
  async atualizar(req, res) {
    try {
      const { id } = req.params;

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

      const {
        nomeCompleto, dataNascimento, genero,
        telefone, telefoneWhatsApp, email,
        cep, logradouro, numero, complemento, bairro, cidad
e, estado
      } = req.body;
```

```
      if (!nomeCompleto) {
        return res.status(400).json({ erro: 'Nome é obrigat
ório' });
      }

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

      if (genero && !['M', 'F', 'O'].includes(genero.toUppe
rCase())) {
        return res.status(400).json({ erro: 'Gênero deve se
r M, F ou O' });
      }

      const dados = {
        nomeCompleto:     nomeCompleto.toUpperCase().trim
(),
        dataNascimento:   dataNascimento || null,
        genero:           genero ? genero.toUpperCase() : n
ull,
        telefone:         telefone ? telefone.replace(/[\s
\-\(\)]/g, '') : null,
        telefoneWhatsApp: telefoneWhatsApp ? 1 : 0,
        email:            email ? email.toUpperCase().trim
() : null,
        cep:              cep ? cep.replace(/[.\-]/g, '') :
null,
        logradouro:       logradouro ? logradouro.toUpperCa
se().trim() : null,
        numero:           numero ? numero.toUpperCase().tri
m() : null,
        complemento:      complemento ? complemento.toUpper
```

```
Case().trim() : null,
        bairro:           bairro ? bairro.toUpperCase().tri
m() : null,
        cidade:           cidade ? cidade.toUpperCase().tri
m() : null,
        estado:           estado ? estado.toUpperCase().tri
m() : null,
      };

      const resultado = await clienteRepository.atualizar(p
arseInt(id), dados);

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

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

    } catch (error) {
      console.error('Erro ao atualizar cliente:', 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 clienteRepository.deletar(par
seInt(id));

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

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

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

}

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

3.3 — `src/routes/clienteRoutes.js`

> **Por quê?** Define todas as rotas da API de clientes com rate limiting de 100 requisições por 15 minutos. A rota de busca `/buscar` vem antes de `/:id` para evitar que o Express interprete a palavra "buscar" como um ID. O middleware `validateJSON` rejeita requisições POST/PUT sem body antes de chegar ao controller.

```
// src/routes/clienteRoutes.js

const express = require('express');
const rateLimit = require('express-rate-limit');
const router = express.Router();
const clienteController = require('../controllers/clienteCo
```

```
ntroller');

/* ============================
   RATE LIMITER
   100 requisições por 15 min
============================ */
const clienteLimiter = 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 obrigatóri
os não fornecidos' });
    }
  }
  next();
};

/* ============================
   ROTAS
   /buscar antes de /:id
   para evitar conflito de parâmetro
============================ */
router.get('/',          clienteLimiter, clienteController.
listarTodos);
router.get('/buscar',    clienteLimiter, clienteController.
buscar);
router.get('/:id',       clienteLimiter, clienteController.
```

```
buscarPorId);
router.post('/',          clienteLimiter, validateJSON, clie
nteController.criar);
router.put('/:id',        clienteLimiter, validateJSON, clie
nteController.atualizar);
router.delete('/:id',     clienteLimiter, clienteController.
deletar);

module.exports = router;
```

FASE 4 — Atualização: server.js

> **Por quê?** O `server.js` precisa registrar as novas rotas de clientes e adicionar a rota estática para a página de clientes no frontend. Apenas duas linhas foram adicionadas ao arquivo existente — o restante permanece idêntico ao AC0.

Linha adicionada no import:

```
const clienteRoutes = require('./src/routes/clienteRoute
s');
```

Linha adicionada no registro de rotas:

```
app.use('/api/clientes', clienteRoutes);
```

FASE 5 — Frontend HTML

5.1 — `public/pages/dashboard.html` (seção clientes adicionada)

> **Por quê?** O dashboard recebeu a seção completa de clientes inline — seguindo o padrão SPA para evitar abertura de múltiplas abas. A seção é colapsável via accordion com 3 painéis: Cadastrar Novo, Buscar/Editar e Listar Todos. Cada painel pode ser aberto e fechado independentemente. Os scripts `clientes.css` e `clientes.js` são carregados no final do documento junto com os demais.

> **Decisão de UX:** Interface SPA (Single Page Application) — toda a interação acontece dentro do dashboard sem abrir novas páginas ou abas, mantendo o contexto do usuário.

Scripts adicionados no `<head>`:

```
<link rel="stylesheet" href="../assets/css/clientes.css" />
```

Scripts adicionados antes do `</body>`:

```
<script src="../assets/js/clientes.js"></script>
```

Estrutura da seção clientes (adicionada dentro do `<main>`):

```
<!-- ============================
     SEÇÃO CLIENTES (SPA)
     Colapsável via accordion
============================ -->
<section id="secao-clientes" class="secao-modulo" style="di
splay:none">
  <h2>Cadastro de Clientes</h2>

  <!-- ACCORDION: CADASTRAR NOVO -->
  <div class="acc-item" id="acc-cadastro">
    <div class="acc-header">
      <span>➕ Cadastrar Novo Cliente</span>
      <span class="acc-arrow">▼</span>
    </div>
    <div class="acc-body">
      <form id="formCliente">
        <input type="hidden" id="clienteId" />

        <div class="form-grid">
          <!-- Tipo -->
          <div class="form-group">
            <label for="tipo">Tipo *</label>
            <select id="tipo" required>
              <option value="">Selecione</option>
              <option value="PF">Pessoa Física</option>
```

```
              <option value="PJ">Pessoa Jurídica</option>
            </select>
          </div>

          <!-- CPF/CNPJ -->
          <div class="form-group">
            <label for="cpfCnpj">CPF / CNPJ *</label>
            <input type="text" id="cpfCnpj" maxlength="18"
required />
          </div>

          <!-- Nome -->
          <div class="form-group form-group-wide">
            <label for="nomeCompleto">Nome Completo / Razão
Social *</label>
            <input type="text" id="nomeCompleto" maxlength
="120" required />
          </div>

          <!-- Data de Nascimento -->
          <div class="form-group">
            <label for="dataNascimento">Data de Nascimento
</label>
            <input type="date" id="dataNascimento" />
          </div>

          <!-- Gênero -->
          <div class="form-group">
            <label for="genero">Gênero</label>
            <select id="genero">
              <option value="">Selecione</option>
              <option value="M">Masculino</option>
              <option value="F">Feminino</option>
              <option value="O">Outro</option>
            </select>
          </div>

          <!-- Telefone -->
```

```
          <div class="form-group">
            <label for="telefone">Telefone</label>
            <input type="text" id="telefone" maxlength="20"
/>
          </div>

          <!-- WhatsApp -->
          <div class="form-group form-group-check">
            <label>
              <input type="checkbox" id="telefoneWhatsApp"
/>
              Este número é WhatsApp
            </label>
          </div>

          <!-- Email -->
          <div class="form-group">
            <label for="email">Email</label>
            <input type="email" id="email" maxlength="254"
/>
          </div>

          <!-- CEP -->
          <div class="form-group">
            <label for="cep">CEP</label>
            <input type="text" id="cep" maxlength="9" />
          </div>

          <!-- Logradouro -->
          <div class="form-group form-group-wide">
            <label for="logradouro">Logradouro</label>
            <input type="text" id="logradouro" maxlength="1
50" />
          </div>

          <!-- Número -->
          <div class="form-group">
            <label for="numero">Número</label>
```

```
            <input type="text" id="numero" maxlength="10" />
          </div>

          <!-- Complemento -->
          <div class="form-group">
            <label for="complemento">Complemento</label>
            <input type="text" id="complemento" maxlength="60" />
          </div>

          <!-- Bairro -->
          <div class="form-group">
            <label for="bairro">Bairro</label>
            <input type="text" id="bairro" maxlength="80" />
          </div>

          <!-- Cidade -->
          <div class="form-group">
            <label for="cidade">Cidade</label>
            <input type="text" id="cidade" maxlength="80" />
          </div>

          <!-- Estado -->
          <div class="form-group">
            <label for="estado">Estado (UF)</label>
            <input type="text" id="estado" maxlength="2" />
          </div>
        </div>

        <div id="formMensagem" class="form-mensagem" style="display:none"></div>

        <div class="form-acoes">
          <button type="submit" class="btn-primary" id="btnSalvar">Salvar Cliente</button>
```

```
          <button type="button" class="btn-secondary" id="b
tnCancelarEdicao" style="display:none">Cancelar Edição</but
ton>
        </div>
      </form>
    </div>
  </div>

  <!-- ACCORDION: BUSCAR / EDITAR -->
  <div class="acc-item" id="acc-busca">
    <div class="acc-header">
      <span>🔍 Buscar / Editar Cliente</span>
      <span class="acc-arrow">▼</span>
    </div>
    <div class="acc-body">
      <div class="busca-row">
        <select id="tipoBusca">
          <option value="nome">Nome</option>
          <option value="cpfcnpj">CPF / CNPJ</option>
          <option value="telefone">Telefone</option>
        </select>
        <input type="text" id="valorBusca" placeholder="Dig
ite para buscar..." />
        <button class="btn-primary" id="btnBuscar">Buscar</
button>
        <button class="btn-secondary" id="btnLimparBusca">L
impar</button>
      </div>

      <div id="resultadoBusca" style="display:none">
        <table class="tabela-clientes">
          <thead>
            <tr>
              <th class="th-ordenavel" id="thNome">Nome ▼</
th>
              <th>CPF/CNPJ</th>
              <th>Telefone</th>
              <th>Ações</th>
```

```
            </tr>
          </thead>
          <tbody id="tbodyBusca"></tbody>
        </table>
      </div>
    </div>
  </div>

  <!-- ACCORDION: LISTAR TODOS -->
  <div class="acc-item" id="acc-lista">
    <div class="acc-header">
      <span>📋 Listar Todos os Clientes</span>
      <span class="acc-arrow">▼</span>
    </div>
    <div class="acc-body">
      <table class="tabela-clientes">
        <thead>
          <tr>
            <th class="th-ordenavel" id="thNomeLista">Nome
▼</th>
            <th>Tipo</th>
            <th>CPF/CNPJ</th>
            <th>Telefone</th>
            <th>Ações</th>
          </tr>
        </thead>
        <tbody id="tbodyClientes"></tbody>
      </table>
    </div>
  </div>

</section>
```

FASE 6 — Frontend CSS

6.1 — `public/assets/css/clientes.css`

> **Por quê?** Estilos exclusivos do módulo de clientes. Define o accordion colapsável (`.acc-item`, `.acc-header`, `.acc-arrow` com rotação animada, `.acc-body` com `display: none/block`), o grid responsivo do formulário (`.form-grid` com `auto-fit`), as badges de gênero coloridas (`.badge-M`, `.badge-F`, `.badge-O`), os botões de ação na tabela (`.btn-editar`, `.btn-excluir`) e o cabeçalho ordenável (`.th-ordenavel`). Responsivo com breakpoints em 768px e 480px.

```
/* ============================
   ACCORDION
============================ */
.acc-item {
  background: #2c3e50;
  border-radius: 8px;
  margin-bottom: 12px;
  overflow: hidden;
}

.acc-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 16px 20px;
  cursor: pointer;
  color: #ecf0f1;
  font-weight: 600;
  font-size: 1rem;
  transition: background 0.2s ease;
}

.acc-header:hover {
  background: #34495e;
}

.acc-arrow {
  transition: transform 0.3s ease;
  font-size: 0.8rem;
}
```

```css
.acc-item.open .acc-arrow {
  transform: rotate(180deg);
}

.acc-body {
  display: none;
  padding: 20px;
  border-top: 1px solid #34495e;
}

.acc-item.open .acc-body {
  display: block;
}

/* ============================
   FORMULÁRIO GRID
============================ */
.form-grid {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(220px, 1f
r));
  gap: 16px;
  margin-bottom: 20px;
}

.form-group-wide {
  grid-column: span 2;
}

.form-group-check {
  display: flex;
  align-items: center;
  padding-top: 28px;
}

.form-group-check label {
  display: flex;
  align-items: center;
```

```
  gap: 8px;
  cursor: pointer;
  color: #ecf0f1;
  font-size: 0.95rem;
}

.form-group-check input[type="checkbox"] {
  width: auto;
  cursor: pointer;
}

/* ============================
   BOTÕES DO FORMULÁRIO
============================ */
.form-acoes {
  display: flex;
  gap: 12px;
  flex-wrap: wrap;
  margin-top: 8px;
}

.btn-primary {
  background: #3498db;
  color: #fff;
  border: none;
  padding: 10px 24px;
  border-radius: 6px;
  font-size: 0.95rem;
  font-weight: 600;
  cursor: pointer;
  transition: background 0.2s ease;
}

.btn-primary:hover {
  background: #2980b9;
}

.btn-secondary {
```

```
  background: #7f8c8d;
  color: #fff;
  border: none;
  padding: 10px 24px;
  border-radius: 6px;
  font-size: 0.95rem;
  font-weight: 600;
  cursor: pointer;
  transition: background 0.2s ease;
}

.btn-secondary:hover {
  background: #636e72;
}

/* ============================
   MENSAGEM DO FORMULÁRIO
============================ */
.form-mensagem {
  padding: 12px 16px;
  border-radius: 6px;
  margin-bottom: 16px;
  font-size: 0.9rem;
  animation: slideIn 0.3s ease;
}

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

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

```css
/* ============================
   BUSCA
============================ */
.busca-row {
  display: flex;
  gap: 12px;
  flex-wrap: wrap;
  margin-bottom: 20px;
  align-items: center;
}

.busca-row select {
  width: auto;
  min-width: 140px;
}

.busca-row input {
  flex: 1;
  min-width: 200px;
}

/* ============================
   TABELA DE CLIENTES
============================ */
.tabela-clientes {
  width: 100%;
  border-collapse: collapse;
  font-size: 0.9rem;
  margin-top: 8px;
}

.tabela-clientes th {
  background: #34495e;
  color: #ecf0f1;
  padding: 10px 14px;
  text-align: left;
  font-weight: 600;
}
```

```css
.tabela-clientes td {
  padding: 10px 14px;
  border-bottom: 1px solid #34495e;
  color: #ecf0f1;
}

.tabela-clientes tr:hover td {
  background: rgba(52, 152, 219, 0.08);
}

/* ============================
   CABEÇALHO ORDENÁVEL
============================ */
.th-ordenavel {
  cursor: pointer;
  user-select: none;
  transition: color 0.2s ease;
}

.th-ordenavel:hover {
  color: #3498db;
}

/* ============================
   BADGES DE GÊNERO
============================ */
.badge-M {
  background: #2980b9;
  color: #fff;
  padding: 2px 8px;
  border-radius: 4px;
  font-size: 0.8rem;
}

.badge-F {
  background: #8e44ad;
  color: #fff;
```

```
  padding: 2px 8px;
  border-radius: 4px;
  font-size: 0.8rem;
}

.badge-O {
  background: #27ae60;
  color: #fff;
  padding: 2px 8px;
  border-radius: 4px;
  font-size: 0.8rem;
}

/* ============================
   BOTÕES DE AÇÃO NA TABELA
============================ */
.btn-editar {
  background: #f39c12;
  color: #fff;
  border: none;
  padding: 5px 12px;
  border-radius: 4px;
  font-size: 0.82rem;
  cursor: pointer;
  margin-right: 6px;
  transition: background 0.2s ease;
}

.btn-editar:hover {
  background: #e67e22;
}

.btn-excluir {
  background: #e74c3c;
  color: #fff;
  border: none;
  padding: 5px 12px;
  border-radius: 4px;
```

```
  font-size: 0.82rem;
  cursor: pointer;
  transition: background 0.2s ease;
}

.btn-excluir:hover {
  background: #c0392b;
}

/* ============================
   RESPONSIVO
============================ */
@media (max-width: 768px) {
  .form-group-wide {
    grid-column: span 1;
  }

  .busca-row {
    flex-direction: column;
  }

  .busca-row select,
  .busca-row input,
  .busca-row button {
    width: 100%;
  }
}

@media (max-width: 480px) {
  .tabela-clientes th,
  .tabela-clientes td {
    padding: 8px 10px;
    font-size: 0.82rem;
  }

  .form-grid {
    grid-template-columns: 1fr;
```

```
  }
}
```

FASE 7 — Frontend JavaScript

7.1 — `public/assets/js/clientes.js`

> **Por quê?** Lógica completa do módulo de clientes. Gerencia o accordion (toggle de painéis), a máscara automática de CPF/CNPJ conforme o tipo selecionado, o envio do formulário (POST para criar ou PUT para editar baseado na presença de `clienteId`), a busca unificada por nome/CPF/telefone, a edição inline (busca dados do cliente, preenche o formulário, abre o painel de cadastro), a exclusão com confirmação, e a listagem completa com ordenação por nome via `localeCompare('pt-BR')`. A tecla Enter aciona a busca automaticamente.

```
// public/assets/js/clientes.js

document.addEventListener('DOMContentLoaded', () => {

  /* ============================
     ACCORDION
     Toggle de abertura/fechamento
  ============================ */
  document.querySelectorAll('.acc-header').forEach((header)
=> {
    header.addEventListener('click', () => {
      const item = header.closest('.acc-item');
      item.classList.toggle('open');
    });
  });

  /* ============================
     ELEMENTOS DO DOM
  ============================ */
  const formCliente      = document.getElementById('formCli
ente');
```

```
  const inputClienteId   = document.getElementById('cliente
Id');
  const inputTipo        = document.getElementById('tipo');
  const inputCpfCnpj     = document.getElementById('cpfCnp
j');
  const inputNome        = document.getElementById('nomeCom
pleto');
  const inputDataNasc    = document.getElementById('dataNas
cimento');
  const inputGenero      = document.getElementById('gener
o');
  const inputTelefone    = document.getElementById('telefon
e');
  const inputWhatsApp    = document.getElementById('telefon
eWhatsApp');
  const inputEmail       = document.getElementById('emai
l');
  const inputCep         = document.getElementById('cep');
  const inputLogradouro  = document.getElementById('logrado
uro');
  const inputNumero      = document.getElementById('numer
o');
  const inputComplemento = document.getElementById('complem
ento');
  const inputBairro      = document.getElementById('bairr
o');
  const inputCidade      = document.getElementById('cidad
e');
  const inputEstado      = document.getElementById('estad
o');
  const divMensagem      = document.getElementById('formMen
sagem');
  const btnSalvar        = document.getElementById('btnSalv
ar');
  const btnCancelarEd    = document.getElementById('btnCanc
elarEdicao');

  const selectTipoBusca  = document.getElementById('tipoBus
```

```
ca');
  const inputValorBusca  = document.getElementById('valorBu
sca');
  const btnBuscar        = document.getElementById('btnBusc
ar');
  const btnLimparBusca   = document.getElementById('btnLimp
arBusca');
  const divResultado     = document.getElementById('resulta
doBusca');
  const tbodyBusca       = document.getElementById('tbodyBu
sca');
  const tbodyClientes    = document.getElementById('tbodyCl
ientes');

  // Variáveis de estado
  let listaClientes = [];
  let ordemNome     = 1; // 1 = A→Z | -1 = Z→A

  /* ============================
     MÁSCARA CPF/CNPJ
     Aplica formatação automática
     conforme tipo selecionado
  ============================ */
  const formatarCpf = (v) => {
    v = v.replace(/\D/g, '').slice(0, 11);
    return v
      .replace(/(\d{3})(\d)/, '$1.$2')
      .replace(/(\d{3})(\d)/, '$1.$2')
      .replace(/(\d{3})(\d{1,2})$/, '$1-$2');
  };

  const formatarCnpj = (v) => {
    v = v.replace(/\D/g, '').slice(0, 14);
    return v
      .replace(/(\d{2})(\d)/, '$1.$2')
      .replace(/(\d{3})(\d)/, '$1.$2')
      .replace(/(\d{3})(\d)/, '$1/$2')
      .replace(/(\d{4})(\d{1,2})$/, '$1-$2');
```

```
  };

  inputCpfCnpj.addEventListener('input', () => {
    const tipo = inputTipo.value;
    if (tipo === 'PF') {
      inputCpfCnpj.value = formatarCpf(inputCpfCnpj.value);
    } else if (tipo === 'PJ') {
      inputCpfCnpj.value = formatarCnpj(inputCpfCnpj.valu
e);
    }
  });

  /* ============================
     MENSAGEM DO FORMULÁRIO
  ============================ */
  const mostrarMensagem = (texto, tipo = 'success') => {
    divMensagem.textContent = texto;
    divMensagem.className = `form-mensagem ${tipo}`;
    divMensagem.style.display = 'block';
    setTimeout(() => { divMensagem.style.display = 'none';
}, 4000);
  };

  /* ============================
     LIMPAR FORMULÁRIO
  ============================ */
  const limparFormulario = () => {
    formCliente.reset();
    inputClienteId.value = '';
    btnSalvar.textContent = 'Salvar Cliente';
    btnCancelarEd.style.display = 'none';
    divMensagem.style.display = 'none';
  };

  /* ============================
     CANCELAR EDIÇÃO
  ============================ */
  btnCancelarEd.addEventListener('click', () => {
```

```
    limparFormulario();
  });

  /* ============================
     SUBMIT DO FORMULÁRIO
     POST (criar) ou PUT (editar)
     baseado na presença de clienteId
  ============================ */
  formCliente.addEventListener('submit', async (e) => {
    e.preventDefault();

    const id = inputClienteId.value;
    const metodo = id ? 'PUT' : 'POST';
    const endpoint = id ? `/clientes/${id}` : '/clientes';

    const dados = {
      tipo:              inputTipo.value,
      cpfCnpj:           inputCpfCnpj.value,
      nomeCompleto:      inputNome.value,
      dataNascimento:    inputDataNasc.value || null,
      genero:            inputGenero.value || null,
      telefone:          inputTelefone.value || null,
      telefoneWhatsApp: inputWhatsApp.checked,
      email:             inputEmail.value || null,
      cep:               inputCep.value || null,
      logradouro:        inputLogradouro.value || null,
      numero:            inputNumero.value || null,
      complemento:       inputComplemento.value || null,
      bairro:            inputBairro.value || null,
      cidade:            inputCidade.value || null,
      estado:            inputEstado.value || null,
    };

    btnSalvar.disabled = true;

    try {
      await apiRequest(endpoint, { method: metodo, body: da
dos });
```

```
      mostrarMensagem(id ? 'Cliente atualizado com sucess
o!' : 'Cliente cadastrado com sucesso!');
      limparFormulario();
      carregarClientes();
    } catch (error) {
      mostrarMensagem(error.message || 'Erro ao salvar clie
nte', 'error');
    } finally {
      btnSalvar.disabled = false;
    }
  });

  /* ============================
     EDITAR CLIENTE
     Busca dados, preenche formulário
     abre painel de cadastro
  ============================ */
  const editarCliente = async (id) => {
    try {
      const cliente = await apiRequest(`/clientes/${id}`);

      inputClienteId.value   = cliente.ClienteId;
      inputTipo.value        = cliente.Tipo;
      inputNome.value        = cliente.NomeCompleto;
      inputDataNasc.value    = cliente.DataNascimento ? cli
ente.DataNascimento.split('T')[0] : '';
      inputGenero.value      = cliente.Genero || '';
      inputTelefone.value    = cliente.Telefone || '';
      inputWhatsApp.checked  = cliente.TelefoneWhatsApp ===
true || cliente.TelefoneWhatsApp === 1;
      inputEmail.value       = cliente.Email || '';
      inputCep.value         = cliente.Cep || '';
      inputLogradouro.value  = cliente.Logradouro || '';
      inputNumero.value      = cliente.Numero || '';
      inputComplemento.value = cliente.Complemento || '';
      inputBairro.value      = cliente.Bairro || '';
      inputCidade.value      = cliente.Cidade || '';
      inputEstado.value      = cliente.Estado || '';
```

```
      // Formata o CPF/CNPJ exibido no campo
      if (cliente.Tipo === 'PF') {
        inputCpfCnpj.value = formatarCpf(cliente.CpfCnpj);
      } else {
        inputCpfCnpj.value = formatarCnpj(cliente.CpfCnpj);
      }

      btnSalvar.textContent = 'Atualizar Cliente';
      btnCancelarEd.style.display = 'inline-block';

      // Abre painel de cadastro, fecha busca
      document.getElementById('acc-cadastro').classList.add
('open');
      document.getElementById('acc-busca').classList.remove
('open');
      limparBusca();

      // Scroll suave até o formulário
      document.getElementById('acc-cadastro').scrollIntoVie
w({ behavior: 'smooth' });

    } catch (error) {
      alert('Erro ao carregar dados do cliente: ' + error.m
essage);
    }
  };

  /* ============================
     EXCLUIR CLIENTE
     Confirm + DELETE + reload
  ============================ */
  const excluirCliente = async (id, nome) => {
    if (!confirm(`Excluir o cliente "${nome}"?`)) return;

    try {
      await apiRequest(`/clientes/${id}`, { method: 'DELET
E' });
```

```
      limparBusca();
      carregarClientes();
    } catch (error) {
      alert('Erro ao excluir cliente: ' + error.message);
    }
  };

  /* ============================
     RENDERIZAR LINHA DA TABELA
  ============================ */
  const renderizarLinha = (c) => {
    const badge = c.Genero
      ? `<span class="badge-${c.Genero}">${c.Genero}</span>
`
      : '—';

    return `
      <tr>
        <td>${c.NomeCompleto} ${badge}</td>
        <td>${c.CpfCnpj || '—'}</td>
        <td>${c.Telefone || '—'}</td>
        <td>
          <button class="btn-editar" onclick="editarCliente
(${c.ClienteId})">Editar</button>
          <button class="btn-excluir" onclick="excluirClien
te(${c.ClienteId}, '${c.NomeCompleto.replace(/'/g,
"\\'")}')">Excluir</button>
        </td>
      </tr>
    `;
  };

  /* ============================
     BUSCAR CLIENTES
  ============================ */
  const buscarClientes = async () => {
    const tipo  = selectTipoBusca.value;
    const valor = inputValorBusca.value.trim();
```

```
    if (!valor) {
      alert('Digite um valor para buscar');
      return;
    }

    try {
      const resultado = await apiRequest(`/clientes/buscar?
tipo=${tipo}&valor=${encodeURIComponent(valor)}`);
      tbodyBusca.innerHTML = resultado.length
        ? resultado.map(renderizarLinha).join('')
        : '<tr><td colspan="4">Nenhum cliente encontrado</t
d></tr>';
      divResultado.style.display = 'block';
    } catch (error) {
      alert('Erro na busca: ' + error.message);
    }
  };

  btnBuscar.addEventListener('click', buscarClientes);

  // Enter aciona busca
  inputValorBusca.addEventListener('keypress', (e) => {
    if (e.key === 'Enter') buscarClientes();
  });

  /* ============================
     LIMPAR BUSCA
  ============================ */
  const limparBusca = () => {
    inputValorBusca.value = '';
    tbodyBusca.innerHTML  = '';
    divResultado.style.display = 'none';
  };

  btnLimparBusca.addEventListener('click', () => {
    limparBusca();
    document.getElementById('acc-busca').classList.remove
```

```
('open');
  });

  /* ============================
     CARREGAR TODOS OS CLIENTES
  ============================ */
  const carregarClientes = async () => {
    try {
      listaClientes = await apiRequest('/clientes');
      renderizarLista();
    } catch (error) {
      tbodyClientes.innerHTML = '<tr><td colspan="5">Erro a
o carregar clientes</td></tr>';
    }
  };

  /* ============================
     RENDERIZAR LISTA COMPLETA
     Respeita ordem atual (ordemNome)
  ============================ */
  const renderizarLista = () => {
    const ordenada = [...listaClientes].sort((a, b) =>
      a.NomeCompleto.localeCompare(b.NomeCompleto, 'pt-BR')
* ordemNome
    );

    tbodyClientes.innerHTML = ordenada.length
      ? ordenada.map((c) => `
          <tr>
            <td>${c.NomeCompleto}</td>
            <td>${c.Tipo}</td>
            <td>${c.CpfCnpj || '—'}</td>
            <td>${c.Telefone || '—'}</td>
            <td>
              <button class="btn-editar" onclick="editarCli
ente(${c.ClienteId})">Editar</button>
              <button class="btn-excluir" onclick="excluirC
liente(${c.ClienteId}, '${c.NomeCompleto.replace(/'/g,
```

```
"\\'")}')">Excluir</button>
            </td>
          </tr>
        `).join('')
      : '<tr><td colspan="5">Nenhum cliente cadastrado</td>
</tr>';
  };

  /* ============================
     ORDENAÇÃO POR NOME
     Clique no cabeçalho alterna A→Z e Z→A
  ============================ */
  const thNomeLista = document.getElementById('thNomeList
a');
  if (thNomeLista) {
    thNomeLista.addEventListener('click', () => {
      ordemNome *= -1;
      thNomeLista.textContent = `Nome ${ordemNome === 1 ?
'▲' : '▼'}`;
      renderizarLista();
    });
  }

  /* ============================
     EXPORTAR FUNÇÕES GLOBALMENTE
     Necessário para onclick inline na tabela
  ============================ */
  window.editarCliente  = editarCliente;
  window.excluirCliente = excluirCliente;

  /* ============================
     INICIALIZAÇÃO
     Carrega lista ao abrir a seção
  ============================ */
  carregarClientes();

});
```

FASE 8 — Checklist de Testes

🗄️ Banco de Dados — SQL Server Management Studio

- [x] Executar `Clientes.sql` — tabela `dbo.Clientes` criada sem erros
- [x] Executar `TesteClientes.sql` — 13 passos sem erros
- [x] `SELECT * FROM dbo.Clientes` retorna registros PF e PJ
- [x] Campo `Ativo = 1` nos registros ativos
- [x] Executar UPDATE e verificar se `DataAtualizacao` mudou automaticamente (trigger)
- [x] Índices `IX_Clientes_NomeCompleto`, `IX_Clientes_CpfCnpj`, `IX_Clientes_Telefone` e `IX_Clientes_Ativo` criados
- [x] Soft delete — `Ativo = 0` após UPDATE, registro permanece na tabela

⚙️ Backend — Postman

> **Pré-requisito:** `node server.js` rodando na porta 3000

- [x] `GET http://127.0.0.1:3000/api/clientes`

Esperado: `200 OK` — array de clientes ativos

- [x] `POST http://127.0.0.1:3000/api/clientes`

Body: `{ "tipo": "PF", "cpfCnpj": "98765432100", "nomeCompleto": "MARIA SILVA", "genero": "F" }`

Esperado: `201 Created`

- [x] `GET /api/clientes/buscar?tipo=nome&valor=maria`

Esperado: `200 OK` — array com cliente encontrado

- [x] `GET /api/clientes/buscar?tipo=cpfcnpj&valor=98765432100`

Esperado: `200 OK` — cliente encontrado

- [x] `GET http://127.0.0.1:3000/api/clientes/1`

Esperado: `200 OK` — objeto do cliente

- [x] `PUT http://127.0.0.1:3000/api/clientes/1`

Body: `{ "nomeCompleto": "MARIA SILVA ATUALIZADA", "genero": "F" }`

Esperado: `200 OK`

- [x] `DELETE http://127.0.0.1:3000/api/clientes/1`

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

- [x] Bug fix: recadastrar CPF com `Ativo = 0` no banco

Esperado: `201 Created` — registro reativado com novos dados

🌐 Frontend — Navegador

> **URL base:** `http://127.0.0.1:3000`

- [x] Login → Dashboard carrega seção Clientes no menu
- [x] Clique em "Cadastro de Clientes" exibe a seção sem abrir nova aba
- [x] Accordion "Cadastrar Novo" abre e fecha corretamente
- [x] Máscara de CPF aplica formatação automática ao digitar (PF)
- [x] Máscara de CNPJ aplica formatação automática ao digitar (PJ)
- [x] Cadastro de novo cliente (PF) — exibe mensagem de sucesso
- [x] Cadastro de novo cliente (PJ) — exibe mensagem de sucesso
- [x] Busca por nome encontra cliente cadastrado
- [x] Busca por CPF/CNPJ encontra cliente cadastrado
- [x] Busca por telefone encontra cliente cadastrado
- [x] Botão Editar preenche formulário com dados do cliente
- [x] Botão Atualizar salva alterações corretamente
- [x] Botão Cancelar Edição limpa formulário
- [x] Botão Excluir exibe confirmação e remove da lista
- [x] Listar Todos exibe todos os clientes ativos
- [x] Clique no cabeçalho "Nome" ordena A→Z e Z→A
- [x] Excluído + recadastrar mesmo CPF → reativa sem erro
- [x] Layout responsivo em tela menor que 768px

FASE 9 — Versionamento GitHub

Passo a Passo — AC1

```
# 1. Entrar na pasta do projeto
cd C:\software-product

# 2. Verificar status — conferir arquivos novos e modificad
os
git status

# 3. Adicionar tudo
git add .

# 4. Verificar se .env não está no stage
git status

# 5. Commit AC1
git commit -m "AC1: módulo cadastro de clientes completo -
CRUD + busca + reativação soft delete"

# 6. Push
git push origin main

# 7. Confirmar no GitHub
# Acessar https://github.com/busellirogerio/software-produc
t
# Verificar se os novos arquivos aparecem:
# - sql/Clientes.sql
# - sql/TesteClientes.sql
# - src/repositories/clienteRepository.js
# - src/controllers/clienteController.js
# - src/routes/clienteRoutes.js
# - public/assets/css/clientes.css
# - public/assets/js/clientes.js
# - public/pages/dashboard.html (atualizado)
# - server.js (atualizado)
```

Arquivos Criados no AC1

Arquivo	Tipo	Descrição
`sql/Clientes.sql`	SQL	Tabela, trigger e índices
`sql/TesteClientes.sql`	SQL	Script de validação (13 passos)
`src/repositories/clienteRepository.js`	Backend	Acesso ao banco — queries SQL
`src/controllers/clienteController.js`	Backend	Validações e respostas HTTP
`src/routes/clienteRoutes.js`	Backend	Rotas da API com rate limiting
`public/assets/css/clientes.css`	Frontend	Estilos do módulo
`public/assets/js/clientes.js`	Frontend	Lógica completa do módulo

Arquivos Atualizados no AC1

Arquivo	O que mudou
`server.js`	Import e registro de `clienteRoutes`
`public/pages/dashboard.html`	Seção clientes + links CSS/JS

AC1 — Versão 1.0 | 19/02/2026 | 9 arquivos criados ou atualizados