📐 Algoritmo Narrativo — Gerenciador de Clientes

Este documento descreve de forma técnica e sequencial o fluxo de execução da aplicação **Gerenciador de Clientes**, que utiliza **Tkinter** para a interface gráfica e **SQLite** como mecanismo de persistência de dados.

---

1. 🚀 Inicialização da Aplicação

- A execução da aplicação se inicia pelo arquivo `application.py`.
- A função principal `main()` é responsável por:
  1. Inicializar o banco de dados chamando `Backend.initDB()`.
  2. Criar a janela principal com `Tk()` (Tkinter).
  3. Instanciar a interface gráfica através da classe `Gui`, passando a janela como argumento.
  4. Iniciar o loop principal da interface (`root.mainloop()`), que mantém a aplicação em execução.

---

2. 🗄️ Inicialização do Banco de Dados

- O método `Backend.initDB()` realiza a configuração do banco de dados:
  - Conecta ao arquivo SQLite `clientes.db`.
  - Executa a instrução SQL para criar a tabela `clientes`, caso ainda não exista, com os seguintes campos:

```
sql
CopiarEditar
id INTEGER PRIMARY KEY AUTOINCREMENT,
nome TEXT NOT NULL,
sobrenome TEXT NOT NULL,
email TEXT NOT NULL,
cpf TEXT NOT NULL
```

  - Aplica `commit` para salvar as alterações e encerra a conexão.

---

3. 📦 Construção da Interface Gráfica

- A classe `Gui` monta a interface utilizando `Tkinter` e `ttk`:
  - Campos de entrada (`Entry`) para: Nome, Sobrenome, Email e CPF.
  - Botões com funções associadas para: Adicionar, Atualizar, Deletar, Buscar e Limpar.
  - Uma tabela (`Treeview`) para listar os clientes cadastrados.

- Ao ser instanciada, a interface carrega automaticamente os registros do banco chamando `Backend.view()`.

---

4. ➕ Adição de Cliente

- Fluxo de ação:
  1. Usuário preenche os campos e clica em "**Adicionar**".
  2. O método verifica se todos os campos estão preenchidos.
  3. Se válido, chama `Backend.insert(nome, sobrenome, email, cpf)`:
     - Insere o registro na tabela `clientes`.
     - Atualiza a interface com os novos dados.
     - Limpa os campos e exibe uma mensagem de confirmação.
  4. Se houver campos vazios, uma mensagem de erro é exibida.

---

5. 📋 Visualização de Clientes

- Sempre que a aplicação é iniciada ou uma operação é realizada, a tabela é atualizada.
- A visualização ocorre via `Backend.view()`, que:
  - Executa `SELECT * FROM clientes`.
  - Retorna todos os registros como lista de tuplas.
  - Preenche o `Treeview` com esses dados.

---

6. 🔍 Busca de Clientes

- Processo de busca:
  1. Usuário insere um ou mais valores nos campos.
  2. Clica no botão "**Buscar**".
  3. A aplicação chama `Backend.search(nome, sobrenome, email, cpf)`, que executa:

     ```
     sql
     CopiarEditar
     SELECT * FROM clientes WHERE
       nome=? OR sobrenome=? OR email=? OR cpf=?
     ```

  4. Os resultados da consulta substituem os dados exibidos na tabela.
  5. Campos em branco são ignorados.

---

7. 🖊 Atualização de Cliente

- Etapas:
    1. Usuário seleciona um cliente na tabela (os dados são carregados nos campos).
    2. Após editar, clica em **"Atualizar"**.
    3. A aplicação verifica se:
        - Todos os campos foram preenchidos.
        - Um cliente foi previamente selecionado (ID válido).
    4. Se válido, chama `Backend.update(id, nome, sobrenome, email, cpf)`:
        - Atualiza os dados no banco de dados.
        - Refresca a tabela.
        - Limpa os campos e exibe confirmação.
    5. Se inválido, exibe uma mensagem de erro.

---

8. ❌ Exclusão de Cliente

- Procedimento:
    1. Usuário seleciona um cliente na tabela.
    2. Clica em **"Deletar"**.
    3. O sistema verifica a seleção e chama `Backend.delete(id)`.
    4. O cliente é removido do banco de dados.
    5. A tabela e os campos são atualizados.
    6. Em caso de não seleção, exibe uma mensagem de erro.

---

9. 🧹 Limpeza de Campos

- Ao clicar em **"Limpar"**, todos os campos de entrada (`Entry`) são redefinidos para vazio.
- Nenhum dado do banco é alterado.

---

10. 🔚 Encerramento da Aplicação

- A aplicação é finalizada quando o usuário fecha a janela gráfica.
- O loop principal do `Tkinter` é encerrado.
- O banco de dados `clientes.db` permanece intacto com os dados salvos.

✅ Resumo Técnico

Componente	Responsabilidade
`application.py`	Ponto de entrada da aplicação; inicializa GUI e banco
`Backend.py`	Operações com banco de dados (CRUD usando SQLite)
`Gui.py`	Interface gráfica e interações com o usuário (Tkinter)
`clientes.db`	Armazenamento persistente local dos registros de clientes

---

Considerações Finais

A aplicação demonstra de forma didática a integração entre:

- **Interface gráfica (GUI)** com **Tkinter**
- **Banco de dados relacional local** com **SQLite**
- **Organização modular** de código em **camadas separadas (GUI vs. Backend)**

Cada evento gerado na interface aciona uma operação no banco de dados, permitindo que os dados permaneçam atualizados, seguros e consistentes.