Resumo de Alterações e Instalações (Excluindo HTML)

Para sua prova, com o frontend HTML/JS pronto, você precisará focar em 3 áreas principais na sua API .NET:

1. **Instalações via NuGet (Entity Framework Core):**
    - Abra o Terminal/CLI (ou Package Manager Console no Visual Studio) na pasta do seu projeto .NET.
    - Instale os pacotes necessários para o Entity Framework Core e SQLite:

      Bash

      ```
      dotnet add package Microsoft.EntityFrameworkCore.Sqlite
      dotnet add package Microsoft.EntityFrameworkCore.Design
      ```

        - `Microsoft.EntityFrameworkCore.Sqlite` : Habilita o uso do SQLite como banco de dados.
        - `Microsoft.EntityFrameworkCore.Design` : Contém ferramentas para criar migrações e atualizar o banco de dados.
2. **Alterações nos Arquivos da sua API .NET:**
    - **`models/Despesas.cs` (Modelo da Despesa):**
        - **Adicione `using System.ComponentModel.DataAnnotations;`** no topo do arquivo.
        - **Adicione `[Key]`** acima da propriedade `id` para marcá-la como chave primária.
        - **Adicione `[Required]`** acima das propriedades `descricao` , `data` , `valor` , `usuarioId` (se quiser que sejam obrigatórias).
        - **Renomeie a propriedade `titulo` para `descricao`** (para corresponder ao `descricao` que vem do seu HTML/JS).
        - **Adicione uma nova propriedade:** `public int usuarioId { get; set; }` (para receber o ID do usuário do frontend).
        - **Ajuste o construtor** (se estiver usando um) para incluir `descricao` e `usuarioId` . Se preferir, pode remover o construtor para que o EF Core

use o padrão sem parâmetros, o que geralmente é mais robusto para deserialização.

- **`appsettings.json` (Configuração da String de Conexão):**
  - Adicione uma seção `ConnectionStrings` com a string de conexão para o SQLite:

    JSON

```
"ConnectionStrings": {
  "DefaultConnection": "Data Source=despesas.db" // Nome do
}
```

- **`Program.cs` (Configuração da Aplicação):**
  - **No início do arquivo, adicione `using Microsoft.EntityFrameworkCore;`** (se não estiver lá).
  - **Configure o DbContext:**

    C#

```
builder.Services.AddDbContext<AppDbContext>(options =>
    options.UseSqlite(builder.Configuration.GetConnectionStr
```

  - **Configure1 o CORS (Cross-Origin Resource Sharing):**
    - Sua política `AllowAll` já está boa para a prova. **Lembre-se do endereço do seu frontend:**

      C#

```
builder.Services.AddCors(options =>
{
    options.AddPolicy(
        "AllowAll", // Nome da política CORS
        builder => builder.AllowAnyOrigin() // Permite
                          .AllowAnyMethod()
                          .AllowAnyHeader()
    );
```

```
});
// Certifique-se de que app.UseCors("AllowAll"); esteja
```

- **Observação Importante para `AllowAnyOrigin()`:** Para um cenário de produção real, você deveria substituir `.AllowAnyOrigin()` por `.WithOrigins("http://localhost:porta_do_seu_frontend", "http://127.0.0.1:porta_do_seu_frontend")` e, se abrir o HTML diretamente, incluir `"null"` temporariamente, para maior segurança. Para a prova, `AllowAnyOrigin()` simplifica.

3. **Executar Migrações do Entity Framework Core:**
   - No Terminal/CLI (na pasta do seu projeto .NET), após fazer as alterações nos arquivos (`Despesas.cs` e `appsettings.json`):
   - **Adicione uma nova migração:**

     Bash

     ```
     dotnet ef migrations add NomeDaSuaMigracaoInicial
     ```

     (Substitua `NomeDaSuaMigracaoInicial` por algo descritivo, ex: `AddDespesaFields`).
   - **Atualize o banco de dados:**

     Bash

     ```
     dotnet ef database update
     ```

     Isso criará ou atualizará o arquivo `despesas.db` e as tabelas com a nova estrutura.

Ao seguir estes passos, sua API .NET estará pronta para receber as requisições POST do seu frontend e interagir com o banco de dados SQLite, cumprindo os requisitos da prova. Lembre-se de rodar sua API e verificar o URL

(`https://localhost:porta_da_sua_api/api/despesas`) e atualizar o `API_URL` no seu JavaScript se necessário.