Documento Técnico

1. Visão geral

A **Operação Curiosidade** permite administrar informações sobre uma pessoa em quatro camadas: **Fatos & Dados**, **Interesses**, **Sentimentos** e **Valores**. O backend expõe endpoints REST para autenticação e gerenciamento de usuários (com suas curiosidades), consumidos pelo front em Angular.

2. Objetivos de engenharia

- Código limpo, legível e modular.
- Camadas separadas (Application/Domain/Infra/UI) para facilitar testes e evolução.
- Segurança básica (JWT) e mitigação de ataques comuns (SQL injection).
- Decisões e trade-offs documentados.

3. Arquitetura (alto nível)

```
graph TD
A[Angular 19 Material] -->|JWT| B[WebApi .NET 8]

B --> C[Application UseCases/DTOs]

C --> D[Domain/Models Entities, Repos Interfaces]

D --> E[Infra EF Core, MySQL, Migrations]

E --> F[MySQL DB]
```

3.1 Pastas/Projetos

- **WebApi**: Controllers, Program, Middlewares mínimos, DI.
- **WebApi.Application**: UseCases, DTOs de request/response, Handlers.
- **WebApi.Models**: Entities, Enums, Interfaces de Repositório, Dtos compartilhados (filtros, paginação).
- **WebApi.Infra**: DbContext, Mappings, Migrations, Implementações de Repositório (EF Core).
- **WebApi.NativeInjector**: Registro de dependências (UseCases/Repos/Services).

4. Domínio & Entidades

4.1 User

- `Id: long`
- `Name: string` *(required, unique pair with email?)*
- `Email: string` *(required, unique)*
- `Age?: int`
- `Address?: string`
- `IsActive: bool`
- Navegação: `Curiosity: Curiosity`

4.2 Curiosity (1:1 User)

- `Id: long`
- `UserId: long`
- `OtherInfo?: string` *(Fatos & Dados)*
- `Interests?: string`
- `Feelings?: string`
- `Values?: string` *(planejado/opcional)*

4.3 Authentication

- `Id`, `UserId`
- `PasswordHash`, `PasswordSalt`
- `LastLoginAt`, `FailedAttempts`, etc.

5. Use Cases (exemplos)

- **ManagerUserUseCase**: cria/atualiza `User` + `Curiosity` a partir de `ManagerUserRequest`.
- **GetUserByIdUseCase**: busca e mapeia para `UserResponse`.
- **GetUserByFilterUseCase**: filtro por nome/email/status, com paginação simples.
- **AuthUseCase (Login)**: valida credenciais, gera JWT.

6. DTOs principais

- **ManagerUserRequest**: `Id?`, `Name*`, `Email*`, `Age?`, `Address?`, `IsActive`, `OtherInfo?`, `Interests?`, `Feelings?`.
- **UserResponse**: espelha o essencial da entidade com objeto `Curiosity` aninhado.
- **GetUserFilterDto**: filtros + paginação.

7. Persistência

- **EF Core** com mapeamentos por Fluent API.
- **Migrations** versionadas; seed inicial via script `init.sql` (usuários de demonstração para login e testes).
- Índices em colunas de busca (ex.: `Email`).

8. Segurança

- **JWT** simples: `Issuer`, `Audience`, `Secret`, `ExpiresInMinutes`.
- **Validações**: DataAnnotations (`[Required]`, `[EmailAddress]`), checagens de unicidade no repositório.
- **SQL Injection**: consultas parametrizadas via EF.

- **Senhas**: hash + salt (ex.: `PBKDF2` / `SHA256` — conforme implementado no projeto). *Nunca* logar senhas.

9. Tratamento de erros & logs

- Controllers retornam `400/404/500` conforme cenário.
- Logs estruturados com `ILogger<T>`; mensagens com contexto (ids, emails).

10. Qualidade & práticas

- **SRP** em UseCases; controllers finos.
- Comentários/XML summaries em DTOs e UseCases.
- Nomes em inglês para código e colunas.

11. Como testar (curl/Insomnia)

11.1 Login

curl -X POST http://localhost:5043/api/auth/login \

-H "Content-Type: application/json" \

-d '{"email":"admin@oc.com","password":"admin"}'

Resposta: `{ token: "..." }`

11.2 Criar usuário

curl -X POST http://localhost:5043/api/user \

-H "Authorization: Bearer <TOKEN>" \

-H "Content-Type: application/json" \

-d '{

"name":"Ana Silva",

"email":"ana@exemplo.com",

"age":28,

"isActive":true,

"otherInfo":"Dados relevantes",

"interests":"Tecnologia",

"feelings":"Motivada"

}'

11.3 Atualizar usuário (atalho do teste)

curl -X POST http://localhost:5043/api/user/1 \

-H "Authorization: Bearer <TOKEN>" \

-H "Content-Type: application/json" \

-d '{

"name":"Ana S. (atualizado)",

"email":"ana@exemplo.com",

"isActive":true

}'

11.4 Buscar por id

curl http://localhost:5043/api/user/1 -H "Authorization: Bearer <TOKEN>"

12. Deploy/Execução

- Local: `dotnet run`.
- Produção: containerização (Docker) opcional; connection string via env var; HTTPS obrigatório.

13. Apêndice — Migrations úteis

criar migration

dotnet ef migrations add InitialCreate

aplicar no banco

dotnet ef database update

reverter última

dotnet ef database update PreviousMigrationName

remover última migration (se ainda não aplicada)

dotnet ef migrations remove

14. Apêndice — Script de seed (`db/init.sql`)

- Cria tabela(s) e insere usuários de exemplo.
- Pode ser usado para popular rapidamente um ambiente de homolog/local.

Contato: Diego Alves Bassualdo — Tech Lead (.NET)

> Este documento pode ser colado no Notion e exportado como PDF. O README acima é mais enxuto; esta seção é o detalhamento para avaliação.

Perfeito! Deixei tudo prontinho em um único documento no canvas:

- **README do backend** (enxuto, direto pro GitHub)
- **Documento técnico detalhado** (pra colar no Notion e exportar em PDF)

Incluí:

- Passo a passo com **MySQL (XAMPP)** → criação do banco → **migrations** ou `init.sql`
- **Connection string** de exemplo e dicas de segurança
- Como **rodar a API** (CLI e Visual Studio)
- **Endpoints** com payloads de criação/edição e respostas
- **Arquitetura por camadas**, fluxo com UseCases/Repos/EF
- **Decisões & trade-offs**, segurança, limitações e próximos passos
- Blocos `curl` para testar no Insomnia/Postman
- Apêndices com comandos EF e seed

Curiosity Dashboard (Angular 19 + Material)

Aplicação **SPA** para gestão de perfis com camadas de informação (Fatos & Dados, Interesses, Sentimentos, Valores). O objetivo é fornecer uma visão consolidada de pessoas, com autenticação e CRUD básico.

> Observação: este repositório contém apenas o front-end. A API REST (.NET) encontra-se em repositório separado. Configure a URL da API em environment.ts.

Índice

- Roadmap
- Licença

Preview

Coloque aqui imagens/gifs do app (login, lista, formulário) quando desejar.

Principais recursos

- **Login com JWT** (validação de campos obrigatórios)
- **Lista de perfis** com tabela, busca e status
- **Formulário de cadastro** com campo **Nome** obrigatório
- **Quatro camadas** (Fatos & Dados, Interesses, Sentimentos, Valores)
- **Interceptor** adicionando `Authorization: Bearer <token>` em chamadas autenticadas
- Layout responsivo com **Angular Material** (toolbar, sidenav, form fields, tabela)

Stack

- **Angular 19** (standalone components + Angular Material)
- **RxJS**
- **Angular Material** (Buttons, FormField, Input, Icon, Toolbar, Sidenav, Table, Tooltip, ProgressSpinner)

Pré-requisitos

- **Node.js 20.x** (LTS recomendado)
- **npm 10+**
- **Angular CLI 19+**:

  npm i -g @angular/cli@19

Começando

1) Clonar e instalar

git clone <url-do-seu-repo-front>

cd <pasta-do-repo-front>

npm ci

> Use npm ci para instalações reproduzíveis (baseado no package-lock.json).

2) Configurar ambientes

Crie/ajuste os arquivos em `src/environments/`:

src/environments/environment.development.ts

/export const environment = {

production: false,

apiBaseUrl: 'http://localhost:5043' // URL local da sua API

```
export const environment = {
production: false,
 apiBaseUrl: 'http://localhost:5043' // URL local da sua API
};
```

src/environments/environment.ts (produção)

export const environment = {

production: true,

apiBaseUrl: 'https://api.seudominio.com' // ajuste para o deploy

};

> Material Icons (ligatures) no index.html (se necessário):

<link rel="stylesheet" href="https://fonts.googleapis.com/icon?family=Material+Icons" />

3) Rodar em desenvolvimento

npm start

ou

ng serve -o

Aplicação disponível em `http://localhost:4200/`.

Scripts NPM

```
{

"start": "ng serve",

"build": "ng build",

"test": "ng test",

"lint": "ng lint",

"format": "prettier --write \"src/**/*.{ts,html,scss}\""

}
```

> Ajuste conforme seu package.json.

Estrutura do projeto

Organização sugerida (pode variar conforme evolução):

src/

app/

core/ # Serviços base, guards, interceptors

interceptors/

auth-token.interceptor.ts

guards/

auth.guard.ts

services/

auth.service.ts

users.service.ts

shared/ # Componentes e utilitários compartilhados

features/

auth/

login/

login.component.ts|html|scss

users/

list/

users-list.component.ts|html|scss

form/

users-form.component.ts|html|scss

app.routes.ts # Definição de rotas

app.component.ts # Shell (toolbar + sidenav)

environments/

environment.ts

environment.development.ts

Integração com API

- **Login**: `POST /api/auth/login` → recebe `{ email, password }` e retorna `{ token }`.
- **Users**:

- `GET /api/user` (filtro)
- `GET /api/user/:id`
- `POST /api/user` (criar/atualizar quando `id` ausente/0)
- `POST /api/user/:id` (atualizar)

Interceptor JWT (resumo):

```
import { HttpInterceptorFn } from '@angular/common/http';
export const authTokenInterceptor: HttpInterceptorFn = (req, next) => {
const token = localStorage.getItem('token');
if (token) {
req = req.clone({
setHeaders: { Authorization: `Bearer ${token}` }
});
}
return next(req);
};
```

Registre no `main.ts` com `provideHttpClient(withInterceptors([authTokenInterceptor]))`.

UI & Tema

- Layout principal: **toolbar** + **sidenav**.
- Componentes principais: Tabela de usuários, formulário com validação (Nome obrigatório), busca com `mat-form-field`.
- Ajustes visuais leves via SCSS (botões *flat/stroked*, ícones, estados de loading com `MatProgressSpinner`).

Acessibilidade

- Labels e `aria-label` em inputs/botões importantes.
- Foco visível; navegação por teclado nas tabelas e formulários.

- `matTooltip` com mensagens sucintas.

Boas práticas de segurança

- Armazene o **token** em `memory` / `sessionStorage` quando possível. Para produção, considere cookies **HttpOnly**.
- Valide dados no backend; trate mensagens de erro genéricas para evitar exposição de detalhes.
- CORS configurado apenas para domínios necessários.

Build & Deploy

- **Build de produção**

  npm run build

  Resultado em `dist/`. Publique o conteúdo no host de sua preferência (S3, Firebase Hosting, Nginx, IIS static, etc.).

- **Ambiente**: ajuste `environment.ts` para apontar a URL correta da API.