ChatGPT

Arquitetura Detalhada: Frontend (Vue.js) & Backend (Microsserviços .NET com Aspire)

Este guia fornece informações detalhadas e passo a passo para embasar o desenvolvimento de ponta a ponta, cobrindo: - Organização dos módulos Aspire - Estrutura de microserviços .NET - Design do banco de dados (ER, migrações, governança) - Arquitetura frontend em Vue.js - CI/CD, segurança, boas práticas e monitoramento

---

1. Organização do Aspire

1.1 Módulos e Convenções

- **Módulo Aspire**: cada domínio (Auth, Church, Member, Content, Event, Finance, Counseling) é implementado como um módulo que herda de `AspireModule`.
- **Assemblies Conventions**: nomear projetos e DLLs como `MyApp.<Domínio>Service` para auto-scan.
- **Registro Automático**: o método `builder.AddAspire(opt => opt.ScanAssemblies(...))` registra controllers, event handlers e middlewares por convenção.

1.2 Pipeline de Inicialização

```
var builder = WebApplication.CreateBuilder(args);
// 1. Adiciona Aspire
builder.AddAspire(opt => opt.ScanAssemblies("MyApp.*Service.dll"));
// 2. Registra módulos individuais (DbContexts, Repositories)
builder.Services.AddDbContext<ChurchDbContext>(...);
builder.Services.AddDbContext<MemberDbContext>(...);
// 3. Identity e JWT via Aspire
builder.Services.AddAspireIdentity(options => { /* config */ });
// 4. Serviços comuns (CORS, HealthChecks, Logging)
builder.Services.AddAspireHealthChecks();
var app = builder.Build();
// 5. Usar Aspire e Static SPA
app.UseAspire();
app.UseStaticFiles();
app.MapSpaStaticFiles();
app.UseSpa(spa => {
  spa.Options.SourcePath = "ClientApp";
  if (app.Environment.IsDevelopment())
spa.UseProxyToSpaDevelopmentServer("http://localhost:5173");
});
app.Run();
```

1.3 Módulos Exemplos

```
public class MemberModule : AspireModule {
  public override void ConfigureServices(IServiceCollection svc) {
    svc.AddDbContext<MemberDbContext>(/* opts */);
    svc.AddScoped<IUserRepository, UserRepository>();
    svc.AddControllers().AddApplicationPart(typeof(MemberModule).Assembly);
  }
}
```

---

2. Microsserviços .NET

2.1 Visão Geral

- **API Gateway**: Ocelot roteando `/auth`, `/church`, `/member`, `/content`, `/event`, `/finance`, `/counseling`.
- **Comunicação Síncrona**: REST + JWT
- **Comunicação Assíncrona**: RabbitMQ via MassTransit integrado nos módulos Aspire

2.2 Contratos e Versionamento

- **OpenAPI/Swagger**: cada módulo expõe seu swagger em `/swagger/v1/swagger.json`
- **Versionamento**: via URL (`/v1/`) e cabeçalhos (`Accept: application/vnd.myapp.v1+json`)

2.3 Padrões de Projeto

- **CQRS com MediatR**: Handlers em `MyApp.<Domain>Service.Application` para comandos e queries
- **Repository + UnitOfWork**: via EF Core DbContext
- **DTOs**: desacoplamento do modelo
- **AutoMapper**: perfis por domínio
- **FluentValidation**: validação de comandos antes de mediatR

2.4 Logging & Observabilidade

- **Serilog**: sink em console, file e Elastic
- **Health Checks**: registrados via `AddAspireHealthChecks()`, expostos em `/health`
- **Tracing**: OpenTelemetry, exportador Jaeger
- **Métricas**: Prometheus exporter

---

3. Modelagem de Dados & Governança

3.1 Diagrama ER Completo (Mermaid)

```
erDiagram
    USUARIO {int id PK varchar nome date nascimento string email}
```

```
    IGREJA {int id PK varchar nome string cnpj text endereco}
    USUARIO ||--o{ IGREJA : "fundou"
    // ... adicione todas as entidades com atributos e ligamentos conforme ER
anterior
```

3.2 Bounded Contexts e Bancos

- Um DbContext e banco por módulo: `AuthDB`, `ChurchDB`, `MemberDB`, etc.
- **Esquemas**: prefixar tabelas para evitar colisões (e.g. `church_patrimonio`)

3.3 Migrations e Deploy

- Incluir projeto `Migrations` dentro de cada módulo
- Scripts CI: ```bash dotnet ef migrations add Initial --project ChurchService.csproj dotnet ef database update --project ChurchService.csproj --connection "${CONN_STR_CHURCH}"

```
- **Rollback**: gerar migração de reversão ou script SQL manual

### 3.4 Qualidade e Auditoria de Dados
- **Fields Auditáveis**: `CreatedBy`, `CreatedAt`, `UpdatedBy`,
`UpdatedAt`, `IsDeleted`, `DeletedAt`
- **Eventos de Domínio**: interceptar `SaveChanges` para publicar
eventos e atualizar réplicas locais
- **Constraints**: FKs, UNIQUE (e.g. `usuario.cpf`), CHECK (e.g.
periodicidade em valores permitidos)
- **Scripts de Governança**: validar registros órfãos ou inconsistentes
via SQL jobs

---

## 4. Banco de Dados: Escolha e Configuração

| Critério           | MySQL                           | SQL
Server                     |
|--------------------|---------------------------------|--------------------------------|
| Driver EF Core     | Pomelo/MySqlConnector           |
Microsoft.EntityFrameworkCore.SqlServer |
| Licença            | Open Source                     | Express (free)/
Paid            |
| Features Avançadas | JSON, Cluster, Particionamento | In-Memory OLTP,
Filestream     |
| Ecosistema .NET    | Bom mas adicional               | Nativo
Microsoft               |

> **Recomendação**: use MySQL se já estiver em produção; escolha SQL
Server em cenários Windows/Azure ou se precisar de OLTP em memória.

---

## 5. Frontend Vue.js Detalhado
```

````
### 5.1 Configuração Inicial
1. `npm init vue@latest client-app`
2. Selecione: TypeScript, Router, Pinia, ESLint+Prettier, Vitest
3. Instale Tailwind:
   ```bash
   npm install -D tailwindcss postcss autoprefixer
   npx tailwindcss init -p
   ```
4. Configure `tailwind.config.js` e importe em `main.ts`.

### 5.2 Composables e Services
- **useAuth**: login, refresh, logout, estado (`isAuthenticated`, `user`)
- **useApiClient(domain)**: retorna instância Axios com baseURL média e interceptors
- **useCrudStore(resource)**: Pinia store genérico para operações CRUD

### 5.3 Componentes Genéricos
- `<BaseInput>`: input + label + erro
- `<BaseModal>`: portal + slots
- `<DataTable>`: table padrão com paginação

### 5.4 Rotas e Guards
```ts
router.beforeEach((to, from) => {
  const { isAuthenticated } = useAuthStore();
  if (to.meta.requiresAuth && !isAuthenticated) return { name:
'Login' };
});
````

5.5 Tratamento de Erros

- **Interceptor**: captura 401, chama refresh, repete requisição
- **Toast Service**: notificações via composable `useToast`

---

6. Segurança e Boas Práticas

- **HTTPS** obrigatório no ASP.NET Core e Vue.js
- **CSP** e headers de segurança (X-Frame-Options, HSTS)
- **Sanitização**: evitar uso indiscriminado de `v-html`
- **Rate Limiting**: via Ocelot ou middleware personalizado
- **Validação no Backend**: FluentValidation em todas as requisições

---

7. CI/CD e Deploy

7.1 Pipeline CI

1. Checkout
````

2. Instalações (`dotnet restore`, `npm install`)
3. Linters e testes (`dotnet test`, `npm run lint`, `npm run test`)
4. Build (`dotnet publish`, `npm run build`)
5. Build Docker Images e push Registry

7.2 Pipeline CD

1. Atualizar migrações e banco (`dotnet ef database update`)
2. Deploy containers no Kubernetes / Azure App Service
3. Smoke tests no ambiente deployado

---

8. Monitoramento e Operações

- **Logs**: Serilog->Elastic, agregados no Kibana
- **Métricas**: Prometheus + Grafana dashboards (latência, erros, throughput)
- **Tracing**: OpenTelemetry spans via Jaeger
- **Alertas**: PagerDuty/Teams/Slack para falhas críticas
- **Backups**: snapshots diários e testes de restore semanal

---

9. Passo a Passo de Adoção

1. Workshop inicial de modelagem e definição de contratos OpenAPI.
2. Scaffold do monorepo com projetos Aspire e Vue.
3. Implementar AuthService e frontend de login.
4. Criar bancos e primeiras migrações.
5. Subir infra (RabbitMQ, banco, API Gateway) em dev local.
6. Desenvolver módulo a módulo: Church → Member → Content → Event.
7. Integrar testes unitários e E2E.
8. Automatizar pipelines CI/CD.
9. Performance tuning e segurança.
10. Go Live e otimize conforme métricas.

Com este documento, você tem um guia exaustivo para arquitetar, codificar, testar e operar todo o sistema de gestão, desde o front-end em Vue até os microsserviços .NET com Aspire, bancos de dados e práticas DevOps.