Modelo C4 do Questionarios

Diagramas Mermaid para o documento tecnico funcional. O codigo fonte de cada diagrama esta nos arquivos `doc/*.mmd` para facilitar edicao e renderizacao.

Como visualizar

- Abra os `.mmd` em um editor com suporte a Mermaid (ex.: VS Code) ou cole o bloco abaixo no mermaid.live.
- Mantenha os arquivos `.mmd` como fonte da verdade; este markdown replica o conteudo para leitura rapida.

C1 - Contexto (quem usa o que)

Fonte: `doc/c4-context.mmd`

```
flowchart LR
    respondent[Respondente Cidadao/participante] -. HTTPS .-> publicSite[Site
Publico\nASP.NET Core MVC]
    analyst[Pesquisador ou Admin Usuario interno] -. HTTPS .-> adminPortal[Portal
Admin\nASP.NET Core MVC]


    adminPortal -->|CRUD de pesquisas e charts| api[API Questionarios\nASP.NET
Core Web API]
    publicSite -->|Obter pesquisa + enviar respostas| api

    api -->|Persistencia| sql[(SQL Server\nEF Core)]
    api -->|Cachear resultados| cache[(Cache Redis ou InMemory)]
    api -->|Publicar respostas| queue[(Fila de mensagens Service Bus ou RabbitMQ)]

    queue --> worker[Worker .NET]
    worker -->|Consumir respostas atualizar agregados| sql
    worker -->|Invalidar/atualizar| cache
    worker -->|Retornar status futuro| api
    api -->|Mensagens de resposta futuro| queue

    subgraph Observabilidade
        obs[Logs / metricas / traces Serilog ou OpenTelemetry]
    end

    api -.-> obs
    worker -.-> obs
    publicSite -.-> obs
    adminPortal -.-> obs
```

Notas: dois atores principais (respondente e analista/admin) acessam front-ends MVC distintos, ambos dependentes da API. A API centraliza persistencia (SQL Server via EF Core), cache de resultados e publicacao

de mensagens para processamento assincrono no worker.

C2 - Container (responsabilidades por deployable)

Fonte: `doc/c4-container.mmd`

```
flowchart TB
    subgraph Frontend
      publicSite[Site Publico\nASP.NET Core MVC/Razor]
      adminPortal[Portal Admin\nASP.NET Core MVC/Razor + Session]
    end

    subgraph Backend
      api[API Questionarios\nASP.NET Core Web API\nControllers + Application
Services]
      worker[Worker de processamento\nBackgroundService]
    end

    subgraph Infra
      sql[(SQL Server\nQuestionariosDbContext + Migrations)]
      cache[(Cache Redis ou MemoryCache\nICacheService)]
      queue[(Fila de mensagens\nIQueueClient)]
      observability[Serilog / OpenTelemetry]
    end

    respondent[Respondente] -->|Responder pesquisa| publicSite
    analyst[Analista/Admin] -->|Criar e acompanhar| adminPortal

    publicSite -->|GET survey / POST responses\nJSON sobre HTTPS| api
    adminPortal -->|CRUD survey, perguntas e opcoes\nJSON sobre HTTPS| api

    api -->|EF Core CRUD| sql
    api -->|Get/Set charts| cache
    api -->|Publicar ResponseMessage| queue

    worker -->|Consumir ResponseMessage\natualizar agregados| sql
    worker -->|Atualizar cache| cache

    api -.-> observability
    worker -.-> observability
    publicSite -.-> observability
    adminPortal -.-> observability
```

Notas: quatro containers .NET (2 MVC, 1 Web API, 1 Worker) comunicam-se via HTTP/JSON e fila. Redis/InMemory e fila sao abstraidos por interfaces. SQL Server e EF Core concentram dados transacionais e agregados.

C3 - Componentes

API Questionarios

Fonte: doc/c4-component-api.mmd

```
flowchart TB
    subgraph API [API Questionarios - Componentes]
      controllers[Controllers REST\nSurvey, Questions, Responses, Results, Users]
      appServices[Application Services\nSurveyService,
QuestionService,\nResponseService, ResultsService, UserService]
      dtos[DTOs / Contracts\nQuestionarios.Application.DTOs]
      domainHelpers[Validacoes de dominio\nEntities + IDateTimeProvider]
      repos[Repositorios EF Core\nSurveyRepository,
QuestionRepository,\nResponseRepository, ResultsRepository, UserRepository]
      cacheSvc[ICacheService\nInMemoryCacheService ou Redis]
      queueSvc[IQueueClient\nConsoleQueueClient ou Service Bus]
      dbCtx[QuestionariosDbContext\nEF Core]
    end

    controllers -->|Mapear HTTP <-> DTOs| appServices
    controllers --> dtos
    appServices -->|Persistencia| repos
    appServices -->|Cache de charts| cacheSvc
    appServices -->|Publicar ResponseMessage| queueSvc
    appServices --> domainHelpers

    repos --> dbCtx
    dbCtx --> sql[(SQL Server)]
    cacheSvc --> cache[(Cache)]
    queueSvc --> queue[(Fila de mensagens)]

    logs[Logs / metricas\nSerilog ou OpenTelemetry] -.-> controllers
    logs -.-> appServices
    logs -.-> repos
```

Notas: controllers convertem HTTP/DTOs e delegam para application services. Services usam repositorios EF Core, cache para charts e fila para respostas. `IDateTimeProvider` facilita testes de regras de tempo. Queue/Cache possuem implementacoes stub que podem ser trocadas.

Portal Admin (MVC)

Fonte: doc/c4-component-portal.mmd

```
flowchart TB
    subgraph PortalAdmin [Portal Admin - Componentes MVC]
        auth[AuthController\nLogin/Logout]
        home[HomeController\nDashboard + charts]
        surveyCtrl[SurveyController\nCRUD de pesquisas]
        questionCtrl[QuestionController\nCadastro de perguntas]
        optionCtrl[OptionController\nCadastro de opcoes]
        views[Razor Views\nLayouts/Partials]
        session[Session state\nUsuario logado + idioma]
        apiClient[QuestionariosApiClient HttpClient\nChama API JSON]
```

```
        models[ViewModels/Models]
    end

    auth --> session
    auth --> apiClient
    surveyCtrl --> apiClient
    questionCtrl --> apiClient
    optionCtrl --> apiClient
    home --> apiClient

    auth --> models
    home --> models
    surveyCtrl --> models
    questionCtrl --> models
    optionCtrl --> models

    models --> views
    session --> views

    apiClient --> api[(API Questionarios)]
    views -->|HTML/Razor| browser[(Browser Admin)]
```

Notas: controllers MVC usam `QuestionariosApiClient` (HttpClient configurado via `ApiOptions`) para chamar a API. Sessao armazena usuario/idioma apos login. Views Razor renderizam os ViewModels. Autenticacao/autorizar ainda e simplificada (sem tokens).

Site Publico (MVC)

Fonte: `doc/c4-component-webpublico.mmd`

```
flowchart TB
    subgraph WebPublico [Site Publico - Componentes MVC]
        surveyController[SurveyController\nExibe survey e envia respostas]
        apiClient[SurveyApiClient HttpClient\nGET survey / POST responses]
        viewModels[ViewModels\nSurveyResponseViewModel\nQuestionAnswerInput]
        views[Razor Views\nForm de respostas]
        settings[ApiSettings\nDefaultSurveyId + BaseUrl]
        logger[Logging\nILogger]
    end

    surveyController --> settings
    surveyController --> viewModels
    surveyController --> apiClient
    viewModels --> views
    apiClient --> api[(API Questionarios)]
    views --> browser[(Respondente no navegador)]
    logger -.-> surveyController
    logger -.-> apiClient
```

Notas: o controller obtem o survey default das `ApiSettings`, monta ViewModel, valida respostas e publica via API. Logger captura falhas de chamada HTTP para alertar operador.

Worker (.NET Background Service)

Fonte: `doc/c4-component-worker.mmd`

```
flowchart TB
    subgraph Worker [Worker .NET - Componentes]
        workerSvc[Worker BackgroundService\nLoop para processar mensagens]
        queueClient[IQueueClient\nSubscribe/Receive ResponseMessage]
        resultsRepo[IResultsRepository\nAtualiza agregados]
        surveyRepo[ISurveyRepository\nConfere survey e perguntas]
        cacheSvc[ICacheService\nInvalidar/atualizar chart]
        clock[IDateTimeProvider]
    end

    workerSvc --> queueClient
    queueClient --> workerSvc
    workerSvc --> resultsRepo
    workerSvc --> surveyRepo
    workerSvc --> cacheSvc
    workerSvc --> clock

    resultsRepo --> sql[(SQL Server)]
    surveyRepo --> sql
    cacheSvc --> cache[(Cache)]

    logs[Logs / metricas\nSerilog ou OpenTelemetry] -.-> workerSvc
```

Notas: o worker projetado consome `ResponseMessage` da fila para atualizar agregados e cache. Hoje o projeto traz apenas o esqueleto do `Worker`, entao o consumo real e uma pendencia tecnica.

Pendencias e proximos passos

- Implementar de fato o consumidor de fila no worker (assinar `IQueueClient`, atualizar agregados e invalidar cache).
- Substituir stubs de cache/fila por Redis/Service Bus reais e incluir health checks.
- Endurecer autenticacao/autorizacao no Portal Admin e eventualmente nos endpoints da API.
- Expandir observabilidade (correlacao de request IDs, traces distribuidos e alertas de latencia/erros).