Documentação Técnica - Task Manager

1. Visão Geral da Solução

Back-end (API) — arquitetura .NET 8, dividida em:

- TaskManagerApi
- TaskManagerApplication
- TaskManagerDomain
- TaskManagerMessaging
- Repository
- Testes automatizados

Front-end (MAUI) — aplicação desktop multiplataforma:

- TaskManagerApp

Integrações:

- SQL Server (via Docker)
- RabbitMQ (mensageria via Docker)

---

2. Arquitetura da Solução

- **Domain-Driven Design (DDD)**: separação de responsabilidades
    - Domain: entidades e contratos
    - Application: comandos e handlers
    - Api: camadas de entrada de dados
    - Repository: acesso a dados

    - Messaging: comunicação assíncrona (RabbitMQ)

- **Padrões**:
    - CQRS com MediatR
    - Injeção de dependências via .NET Core DI
    - Validações com DataAnnotations
    - JWT para autenticação

---

3. Estrutura de Projetos

- **TaskManagerApi**
  API REST (endpoints públicos).

- **TaskManagerApplication**
  Handlers, comandos, validações.

- **TaskManagerDomain**
  Entidades, interfaces de repositórios, enums.

- **Repository**
  Migration para criação de banco e tabelas e implementações de acesso a dados..

- **TaskManagerMessaging**
  Configurações e utilitários de mensageria RabbitMQ.

- **TaskManagerApiIntegrationTests / TaskManagerDomainTest / TaskManagerApplicationTest**
  Testes unitários e testes de integração

- **TaskManagerApp (MAUI)**
  Interface gráfica desktop

---

4. Serviços Docker

- **SQL Server**
    - porta 1433
    - persistência de volume
- **RabbitMQ**
    - porta 5672 (broker)
    - porta 15672 (management UI)
- **taskmanager-api**
    - exposta na porta 7015
    - build Dockerfile

---

5. Endpoints Implementados

User/Login

- **Verbo:** POST
- **Rota:** `/User/Login`
- **Descrição:** Autenticação de usuário retornando JWT.

Requisição esperada:

json:

```
{
  "UserName": "string",
  "Password": "string"
}
```

- **Resposta de sucesso:**
  200 OK + token JWT
- **Possíveis erros:**
  - 404 usuário não encontrado
  - 401 senha incorreta

User/CreateRandom

- **Verbo:** POST
- **Rota:** `/User/CreateRandom`
- **Descrição:** Cria N usuários aleatório. “user_{{random}}”
- **Requisição esperada:**

  json:
  ```
  {
    "amount": number > 0
  }
  ```
- **Possíveis erros:**
  - 201 sucesso, Json com N usuários criados.
  - 401 Unauthorized

User/GetUserById

- **Verbo:** GET
- **Rota:** `/User/GetUserById/{id}`
- **Descrição:** Retorna os dados de um usuário pelo id

- **Resposta esperada:**
  200 OK + objeto User

User/CreateUser

- **Verbo:** POST
- **Rota:** `/User/CreateUser`
- **Descrição:** Cria um novo usuário

Requisição esperada:

json
```
{
  "userName": "string",
  "password": "string"
}
```

- **Resposta de sucesso:**
  201 Created + objeto User

User/GetAllUsers

- **Verbo:** GET
- **Rota:** `/User/GetAllUsers/{withTask}`
- **Descrição:** Lista todos os usuário. De acordo booleano “withTask”, pode ou não retornar as tarefas associadas ao usuário.
- **Resposta esperada:**
  200 OK + lista de User

Task/CreateTask

- **Verbo:** POST
- **Rota:** `/Task/CreateTask`
- **Descrição:** Cria uma nova tarefa. Sempre a tarefa é criada para o usuário logado. (token JWT)

Requisição esperada:

json

```
{
  "data": {
    "title": "string",
    "description": "string",
    "status": 0,
    "userName": "string",
    "dueDate": "2025-08-04T17:10:02.367Z"
  }
}
```

- **Resposta esperada:**
  201 Created

Task/UpdateTask

- **Verbo:** PUT
- **Rota:** `/Task/UpdateTask/{id}`
- **Descrição:** Atualiza dados de uma tarefa
- **Resposta esperada:**
  200 OK

- **Requisição esperada:**

```
json
{
  "data": {
    "title": "string",
    "description": "string",
    "status": 0,
    "userName": "string",
    "dueDate": "2025-08-04T17:10:02.367Z"
  }
}
```

Task/GetTaskById

- **Verbo:** GET
- **Rota:** `/Task/GetTaskById/{id}/{withUser}`
- **Descrição:** Retorna dados de uma tarefa específica por {id}, pode retornar dados do usuário associado a tarefa, caso o parametro boleano {withUser} seja igual a true.

Task/DeleteTask

- **Verbo:** DELETE
- **Rota:** `/Task/DeleteTask/{id}`
- **Descrição:** Remove uma tarefa pelo {id}.

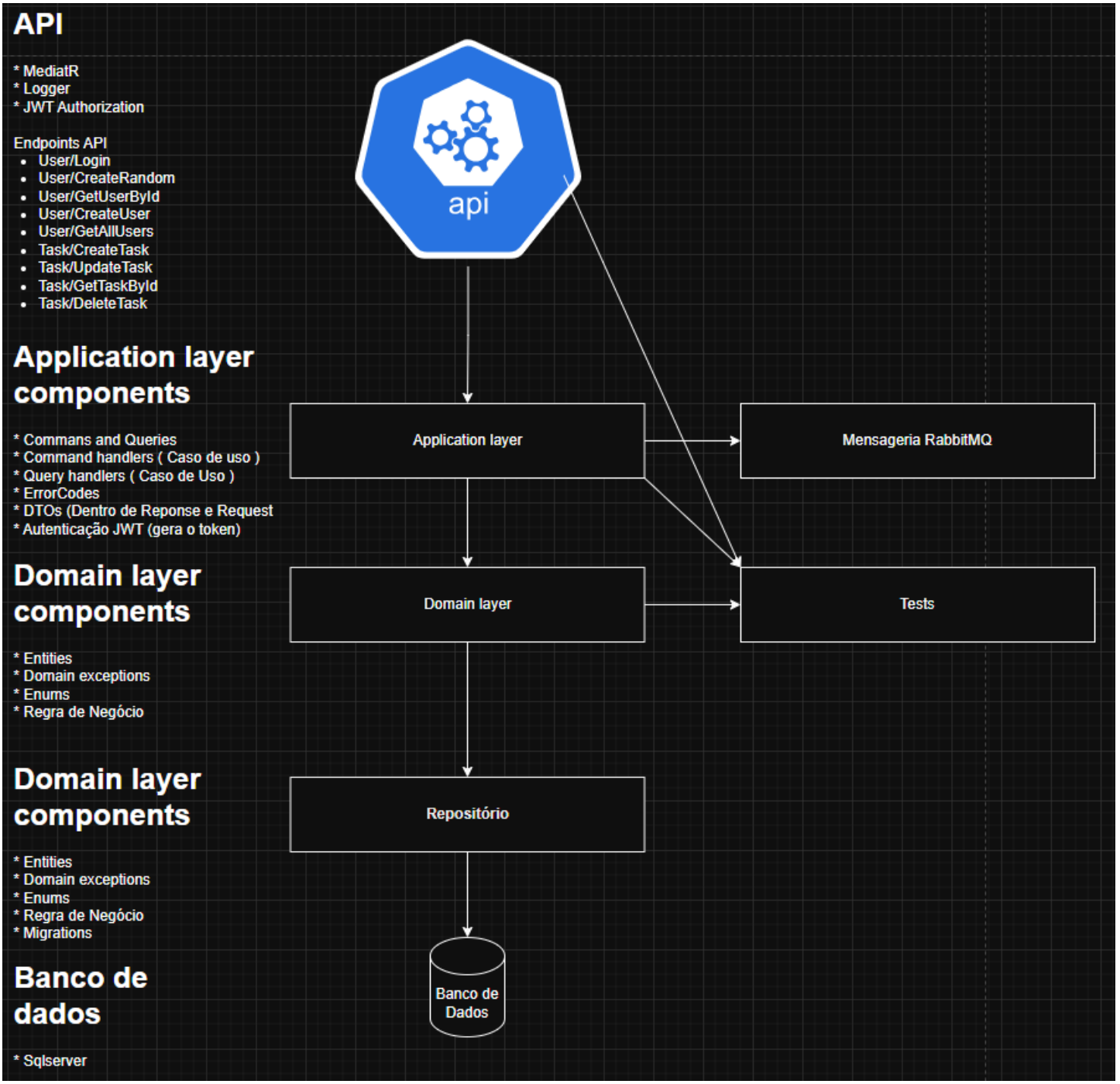
6. Integração com Frontend (TaskManagerApp)

- O front em MAUI consome a API por requisições REST, enviando o Token Bearer para autenticação.

7. Testes

- **TaskManagerApiIntegrationTests**
    - Integração completa simulando requests
- **TaskManagerDomainTest**
    - Valida as regras implementadas a nivel de dominio.
- **TaskManagerApplicationTest**
    - Valida a implementação dos Commands e Queries.

8. Arquitetura UML:



9. Políticas de Segurança JWT

A aplicação está configurada para validar os seguintes aspectos do token JWT:

- Issuer (emissor): só aceita tokens gerados pelo emissor configurado (JWT:Issuer).
- Audience (audiência): só aceita tokens destinados à audiência configurada (JWT:Audience).
- Lifetime (tempo de vida): verifica se o token não está expirado.
- IssuerSigningKey: o token precisa estar assinado com a chave correta (JWT:Key).

appsettings.json:

```
"JWT": {

  "Key": "ZdYM000OLlMQG6VVVp1OH7RxtuEfGvBnXarp7gHuw1qvUC5dcGt3SNM",

  "Issuer": "https://localhost:7066/",

  "Audience": "https://localhost:7066/"

}
```

Expiração do Token

No LoginCommandHandler, o token é gerado com:

expires: DateTime.Now.AddDays(5)

Definindo que o token expira após 5 dias da sua emissão.

Revogação do Token

Atualmente, não há nenhum mecanismo de revogação implementado.

Ou seja, uma vez emitido, o token é válido até sua expiração.

(JWT puro (stateless), pois o token não é armazenado no servidor)

10. Organização de mensagens RabbitMQ

Fila Utilizada:

Nome da fila: `TaskManagerQueue`
Definida via configuração: “appsettings.json”

```
"RabbitMQ": {
  "HostName": "rabbitmq",
  "QueueName": "TaskManagerQueue"
}
```

Essa fila é **criada dinamicamente** pela aplicação, caso não exista, na chamada:

```
await channel.QueueDeclareAsync(
    queue: _queueName,
    durable: false,
    exclusive: false,
    autoDelete: false,
    arguments: null);
```

Envio de Mensagens

Método responsável: NotifyUserAsync(NotificationMessageDto notification)

As mensagens são:

Serializadas em JSON (System.Text.Json)

Publicadas sem exchange (padrão "", ou seja, direct-to-queue)

Usam a própria fila como routingKey:

```
await channel.BasicPublishAsync(
    exchange: "",
    routingKey: _queueName, // "TaskManagerQueue"
    body: body);
```

Topologia

- **Tipo de topologia:** ponto-a-ponto
    - Mensagem enviada diretamente a uma fila (sem fanout, topic ou direct exchange personalizada)
    - **Sem uso de tópicos ou exchanges nomeadas.**

Interface do RabbitMQ

- **Gerenciada via RabbitMQ Management Plugin, acessível em:**

  http://localhost:15672

Resumo:

Item	Valor
Fila principal	TaskManagerQueue
Exchange usada	"" (default exchange)
Routing Key	TaskManagerQueue
Durabilidade da fila	Não (transitória)
Tipo de comunicação	Ponto-a-ponto (direct-to-queue)
Formato da mensagem	JSON
Interface Web	http://localhost:15672