Absolutamente! Aqui está o conteúdo formatado com listas onde apropriado:

**Documentação Funcional do Sistema de Gerenciamento de Tarefas**

**1. Requisitos do CRUD de Tarefas**

**Campos da Tarefa:**

* Título (obrigatório, mínimo 5 caracteres).
* Descrição (opcional, texto livre).
* Status (obrigatório, valores pré-definidos: Aberto, Trabalhando, Concluído).
* Data de Conclusão (opcional, formato ISO 8601).
* Prioridade (obrigatório, valores pré-definidos: Baixa, Média, Alta). (Nota: Adicionado para atender ao filtro de prioridade mencionado no frontend.)
* Sem subtarefas ou relacionamentos.

**2. Autenticação e Usuários**

* Autenticação via JWT.
* Microsserviço de Usuários responsável por:
  + Registro (email, senha).
  + Login (gera token JWT).
  + Cada usuário tem sua lista exclusiva de tarefas (não compartilhadas).
  + Sem hierarquia de usuários (todos são padrão).

**3. Interface do Frontend (Angular 17)**

**Listagem de Tarefas:**

* Filtros:
  + Status (Aberto, Trabalhando, Concluído).
  + Prioridade (Baixa, Média, Alta).
  + Data de conclusão (intervalo).
* Ordenação:
  + Data de criação/conclusão.
  + Prioridade.
* Scroll infinito (paginação automática ao rolar).
* Design responsivo (funciona em mobile/desktop).
* Botão de Atualização Manual para buscar dados em tempo real.
* Exportação para Excel da lista de tarefas.

**4. Arquitetura de Microsserviços**

**Microsserviço de Tarefas:**

* CRUD de tarefas.
* Validações (título, status, prioridade).
* Banco de dados: SQLite (arquivo compartilhado local).

**Microsserviço de Usuários:**

* Autenticação (registro/login).
* Geração de tokens JWT.
* Banco de dados: SQLite (mesmo arquivo compartilhado).
* Comunicação síncrona via REST (HTTP/JSON).
* Atenção: SQLite é adequado apenas para desenvolvimento local. Em produção, substituir por um banco escalável (ex: PostgreSQL) e evitar compartilhamento de arquivo entre microsserviços.

**5. Validações e Tratamento de Erros**

**Validações:**

* Título: obrigatório, mínimo 5 caracteres.
* Status e Prioridade: valores pré-definidos.

**Respostas de Erro:**

* Formato JSON padronizado:

{

"statusCode": 400,

"message": "Título deve ter pelo menos 5 caracteres."

}

* Códigos HTTP claros (ex: 400 para validação, 401 para autenticação).

**6. Integração Frontend/Backend**

* CORS habilitado para todas as origens (configuração do backend).
* Sem upload de arquivos (apenas formulário simples).

**7. Testes e Ambiente**

**Backend (.NET 8):**

* Testes unitários com xUnit ou NUnit.
* Cobertura mínima: validações, autenticação e CRUD.

**Frontend (Angular 17):**

* Testes com Jasmine e Cypress (componentes e fluxos principais).
* Ambiente de execução: Linux local (Docker opcional para simplificar).

**8. Funcionalidades Adicionais**

* Exportação para Excel:
  + Endpoint /tasks/export retorna arquivo .xlsx.
  + Filtros aplicáveis (ex: tarefas concluídas na última semana).
* Sem histórico de alterações ou notificações em tempo real.

**9. Documentação**

* Swagger/OpenAPI:
  + Documentação detalhada dos endpoints (ex: GET /tasks, POST /users/login).
  + Acessível via /swagger no backend.
* Manual de Instalação:
  + Passos para configurar:
    - Backend (restaurar pacotes .NET, executar migrations).
    - Frontend (instalar dependências com npm, configurar URL da API).
  + Exemplo de .env e comandos para Linux.

**Observações Técnicas**

* **Prioridade como Campo Obrigatório:**
  + Adicionado ao modelo de tarefa para atender ao filtro do frontend. Valores: Baixa, Média, Alta.
* **SQLite Compartilhado:**
  + Funcional em ambiente local, mas não recomendado para produção (problemas de concorrência).
* **CORS Aberto:**
  + Restringir origens em produção (ex: apenas domínio do frontend).

Let me know if you would like any other formatting changes!