PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

PUC Minas Virtual

Pós-graduação *Lato Sensu* em Arquitetura de *Software* Distribuído

Projeto Integrado

Relatório Técnico

Sistema para Gestão de Tarefas Pessoais

Ricardo Ferri Capeli

Belo Horizonte
Abri, 2021.0.

Projeto Integrado – Arquitetura de Software Distribuído

Sumário

1. Introdução

Atualmente, a vida moderna é marcada por um ritmo acelerado, onde as responsabilidades pessoais e profissionais competem incessantemente por nossa atenção. Nesse ambiente dinâmico, a eficácia na gestão do tempo e na organização das tarefas pessoais se tornou uma prioridade essencial. Diante desse desafio, a necessidade de ferramentas de gestão de tarefas pessoais, acessíveis e de fácil utilização, tornou-se uma demanda premente.

Neste contexto, nasce o "ProAtividade," uma ferramenta projetada para atender à crescente necessidade de uma solução de gerenciamento de tarefas pessoais que seja intuitiva e prontamente acessível em dispositivos móveis, como smartphones, tablets e laptops. Este trabalho tem como objetivo principal não apenas reconhecer a importância da gestão de tarefas pessoais nas vidas contemporâneas, mas também desenvolver uma solução que proporcione praticidade e eficiência aos usuários.

A ferramenta ProAtividade foi concebida com base em uma abordagem centrada no usuário, enfocando a facilidade de uso como um princípio fundamental. Com uma interface intuitiva e funcionalidades projetadas para simplificar o processo de organização de tarefas, o ProAtividade busca promover a produtividade e o gerenciamento eficaz do tempo.

Este trabalho se propõe a explorar a jornada do desenvolvimento do ProAtividade, desde a concepção da ideia até a sua implementação e disponibilização aos usuários. Além disso, abordaremos as características distintivas que tornam essa ferramenta uma solução eficaz para a gestão de tarefas pessoais, destacando sua capacidade de acesso conveniente em dispositivos móveis

Os objetivos específicos propostos são:

- Desenvolver uma solução de fácil utilização;
- Fácil acesso, por dispositivos moveis como smartphones, tablets e laptops;

- Seja uma solução segura, escalável, tolerante a falhas e robusta para suportar o crescimento no número de acessos.

2. Cronograma do Trabalho

A seguir é apresentado o cronograma proposto para as etapas deste trabalho.

Datas		**Atividade / Tarefa**	**Produto / Resultado**
De	**Até**		
23 / 04 /2023	30 / 04 / 2023	1. Cronograma de trabalho	Construção deste documento.
01 / 05 / 2023	02 / 05 / 2023	2. Contextualização do trabalho	Contextualização deste projeto
07 / 05 / 2023	10 / 05 / 2023	3. Definição de requisitos Arquiteturais	Lista dos requisitos arquiteturais identificados
08 / 05 / 2023	08 / 05 / 2023	4. Definição dos requisitos Funcionais	Lista dos requisitos funcionais identificados
09 / 05 / 2023	09 / 05 / 2023	5. Definição dos requisitos Não-Funcionais	Lista dos requisitos não-funcionais
10 / 05 / 2023	10 / 05 / 2023	6. Definição dos Mecanismos Arquiteturais	Lista dos Mecanismos Arquiteturais identificados
10 / 05 / 2023	30 / 05 / 2023	7. Construção dos Diagramas de Contextos – Modelo C4	Diagrama de contexto criado no Miro e documentado
01 / 06 / 2023	05 / 06 / 2023	8. Revisão da Etapa 1	Documento Etapa 1 revisado
10 / 06 / 2023	08 / 08 / 2023	9. Construção do vídeo de apresentação da Etapa 1	Vídeo concluído da Etapa 1
09 / 08 / 2023	09 / 08 / 2023	10. Apresentação em PPT da Etapa 1	Criação arquivo de apresentação da Etapa 1
09 / 08 / 2023	10 / 08 / 2023	11. Publicação no repositório GitHub Etapa 1	Arquivos criados e disponibilizados no GitHub de forma publica
11 / 10 / 2023	13 / 08 / 2023	12. Construção dos Diagramas de Contêineres	Diagrama de Contêineres
14 / 08 / 2023	16 / 08 / 2023	13. Construção dos Diagramas de Componentes	Diagrama de Componentes
17 / 08 / 2023	25 / 08 / 2023	14. Desenho dos Wireframes da POC	Protótipos de telas de baixa fidelidade
27/ 08/ 2023	15 / 09 / 2023	15. Código da aplicação	Aplicação com 3 requisitos implementados
16 / 09 / 2023	20 / 09 / 2023	16. Publicação código no repositório GitHub Etapa 2	Arquitetos produzidos disponibilizados no GitHub
22 / 09 / 2023	10 / 10 / 2023	17. Análise das abordagens arquiteturais	Documento produzido
11 / 10 / 2023	20 / 10 / 2023	18. Cenários	Documento produzido
21 / 10 / 2023	28 / 10 / 2023	19. Evidências da avaliação	Documento produzido
29 / 10 / 2023	30 / 10 / 2023	20. Resultados obtidos	Documento produzido
30 / 10 / 2023	09 / 11 / 2023	21. Avaliação dos resultados	Documento produzido
30 / 10 / 2023	09 / 11 / 2023	22. Conclusão	Documento produzido

10 / 11 / 2023	10 / 11 / 2023	23. Construção do vídeo de apresentação da Etapa 3	Vídeo da Etapa 3 disponível
10 / 11 / 2023	10 / 11 / 2023	24. Publicação no repositório GitHub Etapa 3	Arquivos disponibilizados no GitHub

3. Especificação Arquitetural da solução

Esta seção apresenta a especificação básica da arquitetura da solução a ser desenvolvida, incluindo diagramas, restrições e requisitos definidos pelo autor, tal que permitem visualizar a macro arquitetura da solução.

3.1 Restrições Arquiteturais

A lista a seguir pontua os requisitos arquiteturais identificados para o desenvolvimento desta solução.

ID	Descrição
RA01	Utilizar as tecnológicas Microsoft para o desenvolvimento parcial da solução.
RA02	Deve ser considerado a nuvem Microsoft Azure como provedora da infraestrutura necessária para a aplicação desenvolvida.
RA03	Deve ser usado a ferramenta Azure DevOps (Boards, Git, CI e CD), para o gerenciamento de todo o ciclo de desenvolvimento e evolução da plataforma.
RA04	A aplicação deve ser acessada pelos principais navegadores como: Google Chrome, Mozilla Firefox e Microsoft Edge.
RA05	A aplicação deve ter uma sessão onde seja possível adicionar as suas tarefas, edita-las e exclui-las.
RA06	A arquitetura deve utilizar o padrão de micro serviços.

3.2 Requisitos Funcionais

Os Requisitos Funcionais listados abaixo são todos que estão associadas as funcionalidades que estabelecem o que o sistema deve fazer.

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF01	A página web deve permitir a realização de um cadastro de uma tarefa.	A	A
RF02	A página web deve abrir um pop-up para inserir as informações.	A	A
RF03	A página web deve permitir adicionar um título.	A	A

RF04	A página web deve permitir selecionar uma prioridade com as opções baixa, normal e alta.	B	A
RF05	A página web deve permitir adicionar uma descrição.	B	A
RF06	A página web deve possuir uma função para salvar as suas informações.	B	A
RF07	A página web deve exibir o número da atividade cadastrada.	B	A
RF08	A página web deve exibir o nome da atividade cadastrada.	B	A
RF09	A página web da aplicação deve exibir a descrição inserida na atividade.	B	A
RF10	A página web da aplicação deve exibir a prioridade classificada por você.	B	A
RF11	Na atividade cadastrada, deve ser apresentado o botão para editar.	B	A
RF12	Ao editar uma atividade existente, deve ser aberto um pop-up com os campos liberados para modificação.	B	A
RF13	No modo editar, deve ser possível alterar o título.	B	A
RF14	No modo editar, deve ser possível alterar a prioridade.	B	A
RF15	No modo editar, deve ser possível alterar a descrição.	B	B
RF16	No modo editar, deve ser possível salvar clicando no botão salvar.	B	B
RF17	No modo editar, deve ser possível cancelar clicando no botão cancelar.	B	A
RF18	Na atividade cadastrada, deve ser apresentado o botão para deletar.	B	A
RF19	Ao clicar no botão deletar em uma das atividades cadastradas, deve ser aberto uma pop-up.	B	A
RF20	No pop-up aberto ao clicar para deletar uma das atividades cadastradas, deve apresentar uma mensagem perguntando se deseja realmente excluir a atividade.	B	A
RF21	No pop-up aberto ao clicar para deletar uma das atividades cadastradas, deve apresentar um botão não onde ao clicar ele volta para a página inicial e não altera em nada a sua atividade.	A	A
RF22	No pop-up aberto ao clicar para deletar uma das atividades cadastradas, deve apresentar um botão sim onde ao acioná-lo será removido aquela atividade.	A	A

*B=Baixa, M=Média, A=Alta.
Obs: acrescente quantas linhas forem necessárias.

3.3 Requisitos Não-funcionais

A lista a seguir apresenta os requisitos não funcionais identificados para o desenvolvimento da aplicação web.

ID	Descrição	Prioridade B/M/A
RNF01	A aplicação deve ser disponibilidade 22 X 7 X 365	A
RNF02	A página web deve suportar uma quantidade de até 10000 cadastro de atividades	A
RNF03	A página web deve ser acessada pelos principais navegadores como Google Chrome, Mozilla Firefox e Microsoft Edge	A
RNF04	A página web da aplicação deve permitir a modificação de qualquer atividade	A
RNF05	O sistema deve ter tolerância a falhas	A
RNF06	O sistema só irá ser acessado e permitir interação para usuários que tenham acesso a internet	A

Obs: acrescente quantas linhas forem necessárias.

3.4 Mecanismos Arquiteturais

Os mecanismos arquiteturais são definidos durante o projeto em três estados:

- Mecanismo de Design;
- Mecanismos de Análise;
- Mecanismos de Implementação.

Análise	*Design*	Implementação
Persistência	ORM	Entity Framework
Persistência	Database	SQLite
Front end		React
Back end	C#	.Net 7
Log do sistema	Telemetria	Azure Monitor Application Insights
Distribuição	Integração e Entrega Continua (CI/CD)	Azure DevOps

4. *Modelagem Arquitetural*

A modelagem arquitetural da solução proposta nesta sessão visa permitir o entendimento da implementação da Prova de Conceito (PoC) da aplicação web na seção 5.

Para esta modelagem arquitetural optou-se por utilizar o modelo C4 para documentação de arquitetura de software. Mais informações a respeito podem ser encontradas aqui: https://c4model.com/ e aqui: https://www.infoq.com/br/articles/C4-architecture-model/. Dos quatro nível que compõem o modelo C4 três serão apresentados aqui e somente o Código será apresentado na próxima seção (5).

4.1 Diagrama de Contexto

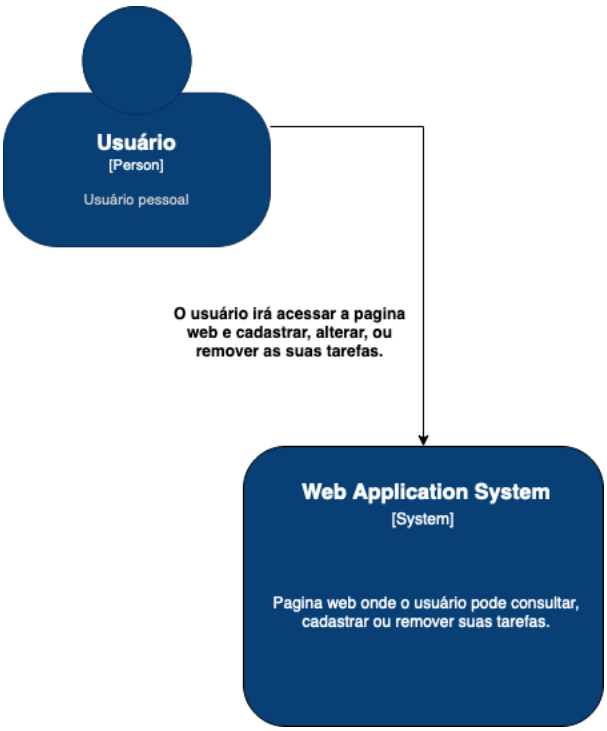

Figura 1 - Visão Geral da Solução

A figura 1 mostra a especificação o diagrama geral da solução proposta, com todos seus principais módulos e suas interfaces.

4.2 Diagrama de Container

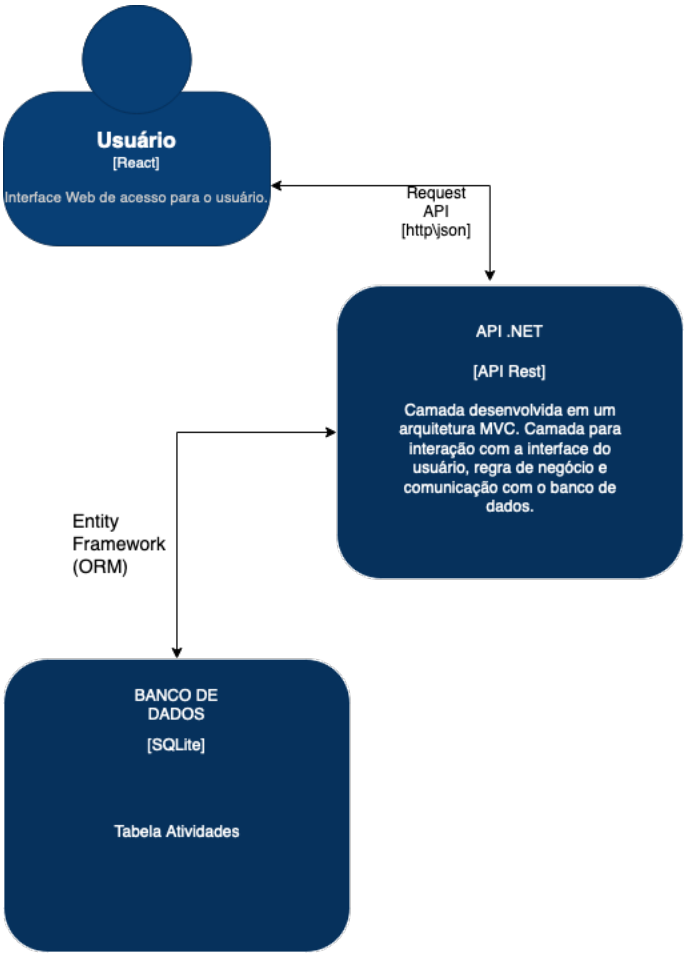

Figura 2 – Diagrama de container

A figura 2 apresenta os containers do Sistema ProAtividade e suas interações com o módulo financeiro.

Nesse diagrama é mostrado como ficará a interação entre o usuário, a página web onde os usuários terão acesso para cadastro, alteração ou remoção de atividades e classificação.

Este diagrama de container ilustra a separação de componentes em camadas: a interface de usuário desenvolvida em React, a API .NET como uma camada intermediária e o banco de dados SQLite como a camada de armazenamento. Essa arquitetura permite que a aplicação seja escalável e facilmente mantida.

A comunicação entre o frontend e cada um dos micros serviços será feita via requisição API Rest/JSON.

Usuário: O usuário acessa a aplicação através de uma interface web desenvolvida em React. Ele interage com a interface para adicionar atividades.

Interface Web (React): A interface do usuário é desenvolvida em React e permite ao usuário adicionar atividades.

API .NET (Serviço Web): A API .NET é um serviço web que recebe as solicitações da interface web, processa as atividades e interage com o banco de dados SQLite para armazenar e recuperar informações sobre as atividades.

Banco de Dados SQLite: O banco de dados SQLite armazena as informações sobre as atividades adicionadas pelos usuários. A API .NET realiza operações de leitura e gravação no banco de dados para manter as atividades.

4.3 Diagrama de Componentes

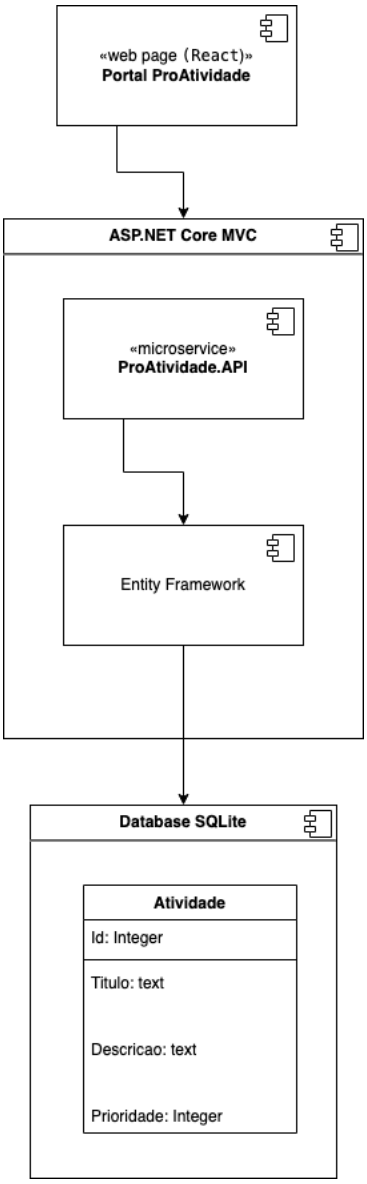

Figura 3 – Diagrama de componentes

Conforme diagrama apresentado na Figura 3, as entidades participantes da solução são:

- Componente Usuário: O usuário interage com a interface web desenvolvida em React para adicionar atividades. Componente API Gateway Ocelot - este componente é responsável por unificar a API e realizar a comunicação com os microserviços.
- Compoente Interface Web (React): A interface do usuário é composta por componentes React que permitem ao usuário adicionar e gerenciar atividades.
- Componentes da API .NET:

- Controlador: O controlador da API .NET recebe solicitações da interface React, roteia as solicitações para o serviço de atividades e envia respostas de volta para a interface.
- Serviço de Atividades: O serviço de atividades contém a lógica de negócios para processar as solicitações, como adicionar atividades ao banco de dados ou recuperar informações sobre atividades existentes.
- Banco de Dados (SQLite): O banco de dados SQLite armazena informações sobre as atividades, permitindo que a API .NET realize operações de leitura e gravação.

5. Prova de Conceito (PoC)

Nessa sessão será detalhada a prova de conceito arquitetural. Para que o objetivo deste trabalho fosse atendido, foram desenvolvidas algumas simulações e foram feitas algumas simplificações negociais, pois o objetivo do trabalho não é validar os requisitos negociais da aplicação, mas sim sua arquitetura.

5.1 Integrações entre Componentes

***Mock Wireframes*:**

Front-end
https://www.figma.com/proto/Q4VpmWEBrcu1OqMa2gMUFC/Atividade?type=design&node-id=1-213&t=2EnULa3ShkU5ewvX-0&scaling=min-zoom&page-id=0%3A1&starting-point-node-id=1%3A213

5.2 Código da Aplicação

Nessa sessão será explicado a nível de código o funcionamento dos requisitos escolhidos. O código fonte completo da aplicação pode ser acessado no endereço:

Codigo: https://github.com/ricardocapeli/tcc-arq-sof-distribuido/tree/main/code
Video: https://github.com/ricardocapeli/tcc-arq-sof-distribuido/blob/main/docs/apresentacaooficial.mp4

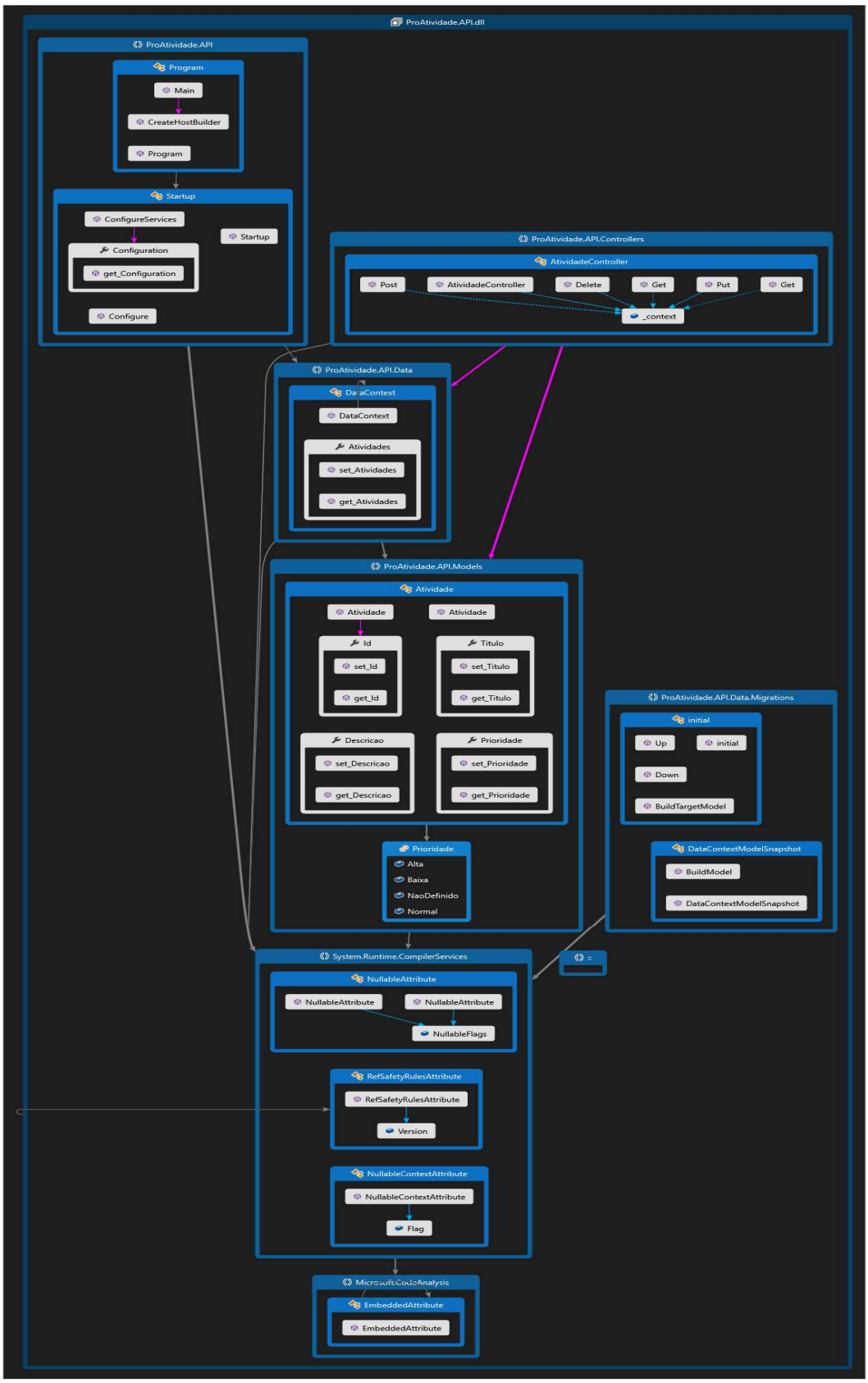
ProAtividade.API.dll
ProAtividade.API
Program
Main
CreateHostBuilder
Program
Startup
ConfigureServices
Startup
Configuration
get_Configuration
Configure
ProAtividade.API.Controllers
AtividadeController
Post
AtividadeController
Delete
Get
Put
Get
_context
ProAtividade.API.Data
DataContext
DataContext
Atividades
set_Atividades
get_Atividades
ProAtividade.API.Models
Atividade
Atividade
Atividade
Id
set_Id
get_Id
Titulo
set_Titulo
get_Titulo
Descricao
set_Descricao
get_Descricao
Prioridade
set_Prioridade
get_Prioridade
Prioridade
Alta
Baixa
NaoDefinido
Normal
ProAtividade.API.Data.Migrations
initial
Up
initial
Down
BuildTargetModel
DataContextModelSnapshot
BuildModel
DataContextModelSnapshot
System.Runtime.CompilerServices
NullableAttribute
NullableAttribute
NullableAttribute
NullableFlags
RefSafetyRulesAttribute
RefSafetyRulesAttribute
Version
NullableContextAttribute
NullableContextAttribute
Flag
Microsoft.CodeAnalysis
EmbeddedAttribute
EmbeddedAttribute

Figura 4 – Estrutura de código da aplicação - ProAtividade

A estrutura da aplicação mostrada na Figura 4 apresenta os componentes de código e suas funções no software implementado:

- DataContext – Esta classe fornece uma conexão lógica entre a aplicação e o banco de dados;
- Configure - Este método configura o NullableAtribute do Entity Framework para garantir o tratamento adequado dos dados;
- OnModelCreating – Este método é usado para definir a forma de suas classes de entidade. Ele define os relacionamentos entre suas classes de entidade e configura como o modelo é mapeado para o banco de dados;
- DataCorrectionTod – Esta classe fornece a metodologia para operações de correção de dados na entidade especificada;
- ApplyDataCorrectionTod – Este método é responsável por invocar o processo de correção de dados.

Operações junto ao banco de dados.

6. Avaliação da Arquitetura (ATAM)

A avaliação da arquitetura desenvolvida neste trabalho é abordada nesta seção visando avaliar se ela atende ao que foi solicitado pelo cliente, segundo o método ATAM.

6.1. Análise das abordagens arquiteturais

Atributos de Quailidade	Cenários	Importância	Complexidade
Usabilidade	Cenário 1: O sistema deve permitir o cadastro de novas atividades.	A	M
Usabilidade	Cenário 2: O sistema deve prover boa usabilidade e compatibilidade browsers distintos.	A	B
Usabilidade	Cenário 3: O sistema deve permitir deletar atividades existentes.	A	M

Usabilidade	Cenário 4: O sistema deve ser responsável.	M	B
Usabilidade	Cenário 5: O sistema deve permitir a edição de atividades cadastradas.	A	M

6.2. Cenários

Cenário 1 - Usabilidade: Ao acessar o sistema ele deve permitir o cadastro de novas atividades através do botão "+".

Cenário 2 - Usabilidade: A navegação pela interface do sistema deve ser simples e intuitiva, com nomes e ícones de fácil percepção para ajudar no entendimento da navegação independente do browser utilizado: Google Chrome, Mozila Firefox, Microsoft Edge, entre outros.

Cenário 3 – Usabilidade: O sistema deve permitir deletar as atividades existentes através do botão deletar.

Cenário 4 – Usabilidade: Ao navegar pela interface utilizando um navegador móvel, deve ser possível executar todas as funções do sistema, de maneira responsiva.

Cenário 5 – Usabilidade: O sistema deve permitir a edição das atividades existentes através do botão "editar".

6.3. Evidências da Avaliação

Atributo de Qualidade:	Usabilidade
Requisito de Qualidade:	O sistema deve permitir o cadastro de novas atividades.
Preocupação:	
Ao acessar o sistema ele deve permitir o cadastro de novas atividades através do botão "+".	
Cenário(s):	
Cenário 1	
Ambiente:	
Sistema em operação normal	

Estímulo:	
Ao clicar no botão + no canto superior direito ele deve abrir uma pop-up para cadastro de nova atividade.	
Mecanismo:	
Criar um serviço REST para atender às requisições do sistema de monitoramento	
Medida de resposta:	
Retornar os dados requisitados no formato JSON	
Considerações sobre a arquitetura:	
Riscos:	Alguma instabilidade na rede pode deixar a conexão lenta ou mesmo a perda de pacotes.
Pontos de Sensibilidade:	Não há
Tradeoff:	Não há

6.4. Resultados Obtidos

Após a avaliação dos critérios de qualidade propostos, foi constatado que o objetivo foi alcançado. Os resultados obtidos para os atributos de disponibilidade, usabilidade, desempenho, rastreabilidade, segurança e acessibilidade estão apresentados em uma tabela a seguir:

Requisitos Não Funcionais	Teste	Homologação
RNF01: O sistema deve ter o design responsivo	OK	OK
RNF02: O sistema deve ser responsivo web	OK	OK
RNF03: O sistema deve ser responsivo para todos os tipos de tela (Computadores, Notebooks, Tablets e Smartphones).	OK	OK

7. Avaliação Crítica dos Resultados

Uma avaliação crítica do resultado da aplicação descrita anteriormente, que inclui uma interface React, uma API .NET e um banco de dados SQLite para gerenciar atividades, pode ajudar a entender melhor os pontos fortes e as áreas de melhoria do sistema.

Após uma análise abrangente da arquitetura deste projeto, destacam-se dois aspectos como os mais cruciais: segurança e facilidade de uso. Considerando esses critérios e o orçamento disponível, a arquitetura proposta atende satisfatoriamente aos requisitos estabelecidos.

A escolha de implementar a interface gráfica com ASP.NET Core 6.0 revelou-se vantajosa no desenvolvimento, pois oferece robustez, segurança e facilidade de implementação, aproveitando as capacidades da linguagem C# e as facilidades fornecidas pelo framework. A abordagem de dividir o projeto em módulos independentes, utilizando o padrão MVC e outras decisões de segregação de responsabilidades, contribuiu para a solidez da arquitetura, simplificando o gerenciamento e a manutenção.

A estratégia em camadas também oferece a flexibilidade de, no futuro, migrar a camada de apresentação para outras tecnologias, como React, Angular ou qualquer outro framework de front-end que seja mais adequado. Além disso, a escolha de utilizar bibliotecas como o Bootstrap para o desenvolvimento dos layouts revelou-se acertada. Isso se deve à variedade de recursos e classes nativas disponíveis na biblioteca, que resultaram em um design limpo, simples e de fácil utilização.

8. Conclusão

Durante o desenvolvimento deste projeto, foram extraídas diversas lições valiosas. A elaboração de uma arquitetura sólida demanda planejamento criterioso e uma curva de aprendizado significativa. No entanto, quando essa arquitetura é adequadamente configurada e alinhada com os objetivos do negócio, as mudanças nela se tornam mais evidentes e simplificadas, graças à sua compreensão facilitada.

É essencial que a arquitetura esteja em consonância constante com o negócio, pois somente assim poderá corresponder às expectativas estabelecidas. Dentre as lições aprendidas, destacam-se as seguintes:

Ter uma arquitetura bem definida e alinhada com o negócio minimiza retrabalhos em caso de mudanças, resultando em redução de custos. Isso ocorre porque a arquitetura se harmoniza com as expectativas da empresa.

A criação de um site responsivo gera maior tráfego e retém um número maior de usuários. A facilidade, rapidez e fluidez na utilização das funcionalidades aprimoram significativamente a usabilidade.

Além disso, foram identificadas oportunidades de melhoria. Por exemplo, no momento, o batch scheduler realiza leituras frequentes no banco de dados em busca de novas mensagens a serem disparadas. Em vez disso, é possível substituir esse scheduler por um lambda, uma opção oferecida por diversos serviços de nuvem, simplificando o processo de envio de mensagens com custos reduzidos. A arquitetura também possibilita a adição de novos serviços, como o envio de notificações push, mantendo-se flexível e adaptável às necessidades em constante evolução.