MComics

Documento de Arquitetura

Histórico da Revisão do Documento

Data	Versão	Páginas modificadas	Autor(es)
26/04/2021	1.0	01-24	Jacob, Abigail

1. Introdução

1.1. Finalidade

A finalidade deste documento é referenciar os artefatos arquiteturais da aplicação web MComics. Ele apresenta um aspecto geral abrangente da arquitetura e utiliza as diversas visões das seções a seguir para demonstrar diferentes perspectivas do sistema, com o objetivo de registrar as decisões mais importantes tomadas pela equipe do software para seu desenvolvimento.

1.2. Escopo

Este documento apresenta a visão arquitetural do MComics e não aborda especificidades referentes às tecnologias escolhidas para sua implementação.

1.3. Visão geral

A definição arquitetural deste projeto de desenvolvimento do software MComics segue as documentações criadas até o presente momento, que guiaram a primeira versão (e devem guiar as versões subsequentes), visto que a finalidade do documento é estabelecer os fundamentos arquiteturais da aplicação.

1.4. Definições, acrônimos e abreviações

.NET Core 5.0: é um framework livre e de código aberto para os sistemas operacionais Windows, Linux e macOS. O projeto é desenvolvido principalmente pela Microsoft.

3-Tier Architecture: arquitetura que organiza os aplicativos em três camadas de computação lógica e física: a camada de apresentação ou interface do usuário, a camada da aplicação, onde os dados são processados, e a camada de dados, onde os dados associados ao aplicativo são armazenados e gerenciados.

Azure: é uma plataforma destinada à execução de aplicativos e serviços, baseada nos conceitos da computação em nuvem.

Cliente-servidor: segundo Sommerville (2016), é o estilo arquitetural cliente-servidor é aquele no qual um aplicativo é modelado como um conjunto de serviços fornecidos por servidores. Os clientes podem acessar esses serviços e

apresentar os resultados aos usuários finais. Clientes e servidores são processos separados.

CQRS: significa Command Query Responsibility Segregation. É um padrão arquitetural sobre separar a responsabilidade de escrita e leitura de seus dados.

Entity Framework 5.0: é um mapeador moderno de banco de dados de objeto para .NET. Ele dá suporte a consultas LINQ, controle de alterações, atualizações e migrações de esquema.

Front-end: é a prática de converter dados em uma interface gráfica, (neste caso) através do uso de HTML, CSS e JavaScript, para que os usuários possam visualizar e interagir com esses dados.

Generics Pattern: é um padrão de projeto que define a criação de interfaces genéricas únicas, para qualquer tipo de dado ou entidade, associando-o com cada componente importante do sistema, separados por contextos, facilitando a injeção de dependendência e reaproveitamento de código.

HTTPS: é uma implementação do protocolo HTTP sobre uma camada adicional de segurança que utiliza o protocolo SSL/TLS.

Identity Server 4: é um framework que implementa os protocolos OpenID Connect e OAuth 2.0 para o NET Core.

JWT: JSON Web Token é um padrão da Internet para a criação de dados com assinatura opcional e/ou criptografia cujo payload contém o JSON que afirma algum número de declarações. Os tokens são assinados usando um segredo privado ou uma chave pública/privada.

React: é uma biblioteca JavaScript para construção de interfaces de usuário.

Repository Pattern: é um padrão de projeto que permite um encapsulamento da lógica de acesso a dados, impulsionando o uso da injeção de dependência (DI) e proporcionando uma visão mais orientada a objetos das interações com o banco de dados.

SPA Pattern: é um padrão arquitetural de um aplicativo da web ou site que interage com o usuário reescrevendo dinamicamente a página da web atual com novos

dados do servidor. O objetivo são transições mais rápidas que façam o site parecer mais um aplicativo nativo.

SQL Server: é um sistema gerenciador de banco de dados relacional.

Swagger: é uma linguagem de descrição de interface para descrever APIs RESTful expressas usando JSON.

2. Representação arquitetural

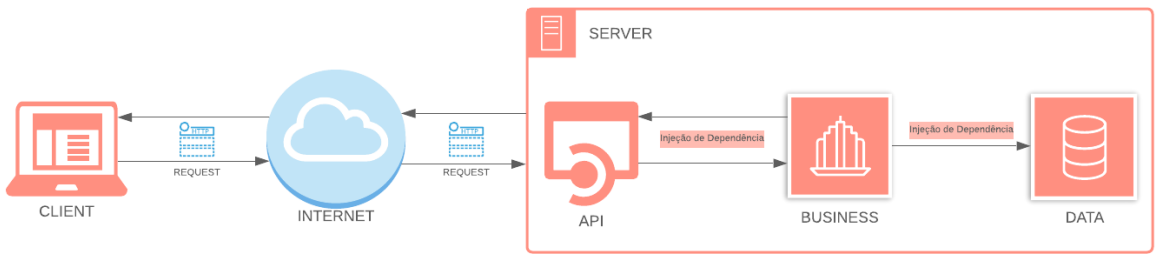
A representação geral da arquitetura demonstra a modelagem arquitetônica utilizando o estilo arquitetural Cliente-Servidor, em conjunto com os padrões arquiteturais 3-Tier Architecture, CQRS, SPA Pattern.

Na representação arquitetural apresentada na Figura 1 temos a aplicação dividida em duas frentes: o “Client”, que renderiza e hospeda a camada de visualização em seu próprio navegador realizando conexões via HTTPS com o servidor e, em contrapartida, o “Server” é hospedado em um servidor de nuvem da Azure, que recebe as requisições via HTTPS e, através das injeções de dependência, se conecta com a camada de “Business” da aplicação, realizando toda a lógica de negócio das funcionalidades necessárias. Por fim, persiste os dados na camada de “Data” integrando com o banco relacional na nuvem.

2.1. Diagrama da visão geral da representação arquitetural

O diagrama abaixo, assim como os demais que estarão neste documento, podem ser visualizados em qualidade e tamanho melhores no repositório do software no GitHub.

Figura 1 - Visão geral da representação arquitetural



Fonte: Elaborada pelos autores, 2021.

3. Metas e restrições da arquitetura

Esta seção descreve as metas do software que têm algum impacto na arquitetura. Também são descritas as restrições arquiteturais que se aplicam ao projeto, tais como: estratégias de modelagem, de implementação e as ferramentas de desenvolvimento.

Tabela 1 - Restrições arquiteturais

Boas práticas	O sistema será desenvolvido utilizando os conceitos de Programação Orientada a Objetos, SOLID, DDD, Heurísticas de Nielsen, Clean Code, boas práticas de UX e de Engenharia de Software.
Frameworks	O sistema utilizará os principais frameworks do mercado para o desenvolvimento de aplicações Web: React, .NET Core 5.0, Entity Framework 5.0, Identity Server 4, Swagger, Azure Host, SQL Server, JWT e HTTPS.
Linguagens de programação	As linguagens adotadas para o desenvolvimento do sistema foram: TypeScript, C#, SQL, HTML, SCSS.
Padrões arquiteturais e padrões de projeto	Foi adotado para o projeto os seguintes padrões arquitetônicos e padrões de projeto: CQRS, SPA Pattern, Repository Pattern e Generics Pattern.

Fonte: Elaborada pelos autores, 2021.

4. Visão de casos de uso

O diagrama da Figura 2 abaixo faz alusão a todos os casos de uso que o software MComics atende.

Figura 2 - Diagrama de casos de uso

Fonte: Arruda e Ferraz (2021)[1].

4.1. Realizações de casos de uso

As realizações dos casos de uso, cujo diagrama está representado acima, estão sendo ilustradas abaixo através de suas descrições.

4.1.1. Fazer login no sistema

Esse caso de uso ocorre quando um usuário já possui uma conta no MComics e deseja logar-se. Para isso ele deve informar o email e senha que foram utilizados em seu cadastro.

4.1.2. Criar conta no sistema

Esse caso de uso ocorre quando o usuário deseja criar uma conta para utilizar as funções do MComics. Para isso, ele precisa preencher os campos disponíveis com seu nome, email e senha. Após o cadastro ele receberá um email para confirmar sua conta e em seguida, caso confirme, ele poderá fazer login no sistema.

4.1.3. Escolher a imagem de perfil baseada nos personagens

Esse caso de uso ocorre quando o usuário deseja alterar sua imagem de perfil, seja ao criar sua conta ou posteriormente. Para escolher uma miniatura de um dos personagens da Marvel ele deve buscar pelo nome do personagem e selecionar a miniatura para atribuí-la a seu perfil.

4.1.4. Avaliar um quadrinho

Esse caso de uso ocorre quando o usuário dá uma nota para um quadrinho selecionado por ele. Para isso ele deve selecionar um quadrinho, selecionar o campo de notas e informar uma avaliação entre 0.5 e 5.

4.1.5. Marcar um quadrinho como “lido”

Esse caso de uso ocorre quando o usuário deseja adicionar um quadrinho a sua lista de quadrinhos lidos. O usuário deve estar cadastrado, autenticado e logado no MComics, estar na página de um quadrinho e selecionar a opção “lido”.

4.1.6. Realizar buscas por personagens, quadrinhos e eventos

Esse caso de uso ocorre quando o usuário realiza buscas no software por personagens, quadrinhos e eventos. Para isso ele deve informar o nome da informação desejada e selecionar o resultado de seu interesse.

4.1.7. Gerenciar listas de interesses em seu perfil

Esse caso de uso ocorre quando o usuário deseja visualizar suas listas de quadrinhos marcados como “lido” ou como “pretendo ler”. Para isso ele deve abrir sua página de perfil e selecionar a lista que quiser.

4.1.8. Marcar quadrinho como “pretendo ler”

Esse caso de uso ocorre quando o usuário deseja adicionar um quadrinho a sua lista de quadrinhos para ler futuramente. O usuário deve: estar cadastrado, autenticado e logado no MComics, estar na página de um quadrinho e selecionar a opção “pretendo ler”.

4.1.9. Comentar em um quadrinho

Esse caso de uso ocorre quando o usuário deseja fazer um comentário em um quadrinho. Ele deve estar cadastrado, autenticado e logado no MComics. Em seguida, ele deve buscar e selecionar um quadrinho, acessando a página escolhida. Nela ele deve selecionar a funcionalidade para comentar, escrever e publicar seu comentário.

4.1.10. Disponibilizar dados de personagens, quadrinhos e eventos

Esse caso de uso ocorre quando a API da Marvel permite que um serviço faça uma requisição aos seus dados, disponibilizando personagens, quadrinhos e eventos em seu site. Para isso ela permite que requisições através de métodos HTTP e retorna os dados em formato JSON.

5. Visão lógica

Este tópico apresenta a visão lógica da arquitetura, demonstrando a perspectiva organizacional e estrutural dos componentes e seus relacionamentos. Segundo o estilo arquitetônico de cliente servidor, a aplicação foi dividida em duas frontes, o “Client” que será composto por apenas um módulo, adotando um estilo arquitetural monolítico e o “Server” em contrapartida, implementa o estilo arquitetural 3-Tier Architecture, que o divide em três camadas físicas, além de um conjunto de padrões arquitetônicos descritos no tópico 2.

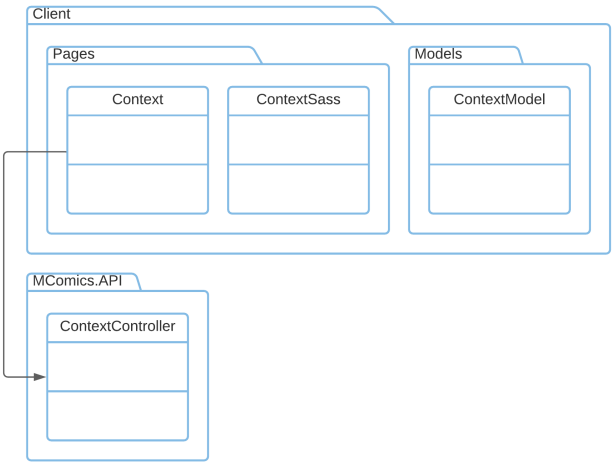
A comunicação entre o “Client” e o “Server” será por meio dos métodos do protocolo HTTP, e a utilização do padrão REST, internamente, a comunicação entre as camadas do “Server” será por meio de injeção de dependências, como boa prática de engenharia de software. A representação nos diagramas apontados abaixo, especifica a relação entre o “Client” e o “Server”, especificando os principais elementos do projeto, suas interfaces e relacionamentos internos e externos.

5.1. Pacotes de design significativos do ponto de vista da arquitetura

5.1.1. Client

Camada responsável pela apresentação e pela lógica da apresentação e por solicitar serviços ao servidor. É a parte que interage com o usuário, comumente chamado de front-end da aplicação.

Figura 3 - Representação arquitetural da camada Client

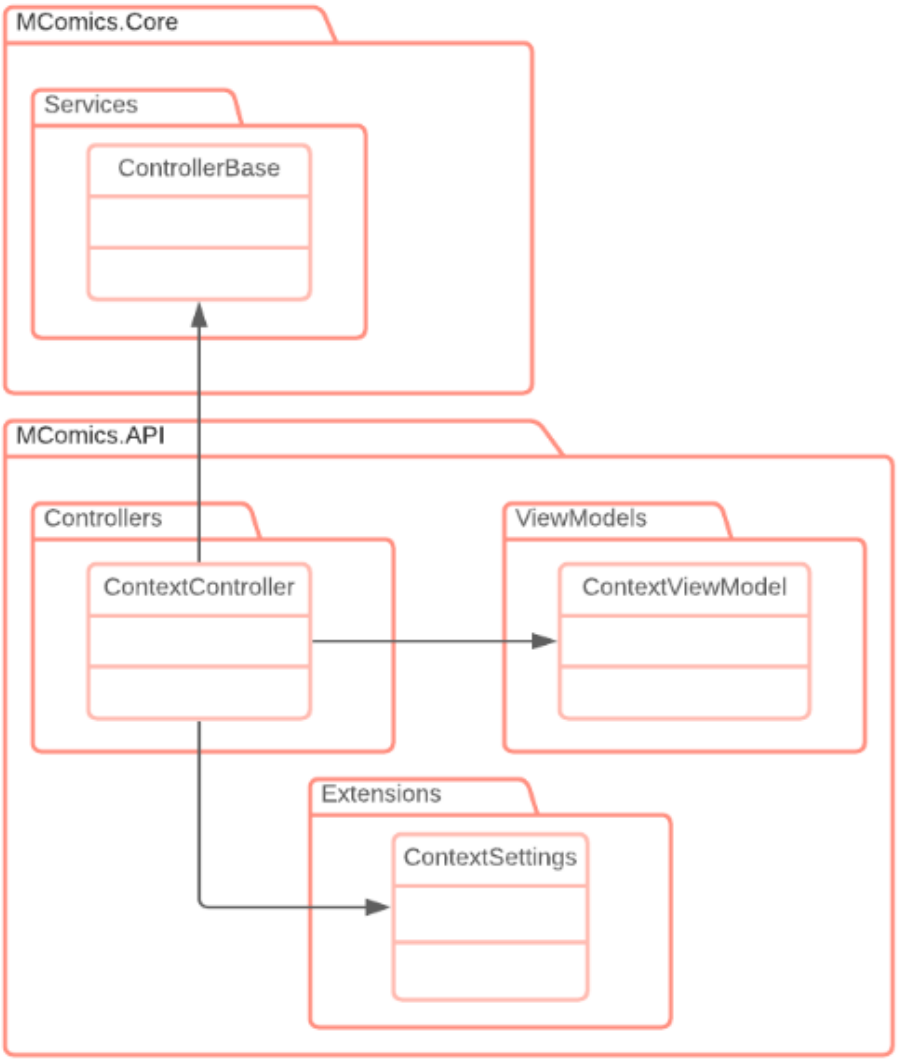


Fonte: Elaborada pelos autores, 2021.

5.1.2. Server: API

Camada responsável por controlar e gerenciar as entradas e saídas do servidor, repassando os dados formatados em entidades de apresentação ao usuário, estando no mais alto nível do negócio.

Figura 4 - Representação arquitetural da camada API do servidor

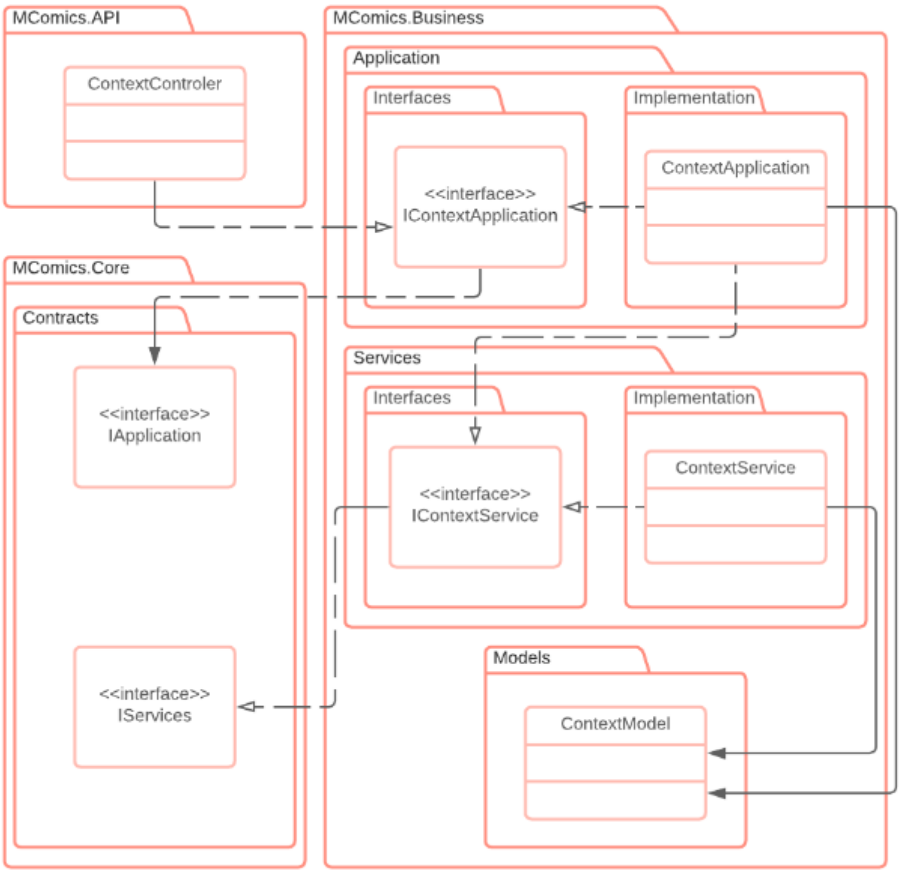


Fonte: Elaborada pelos autores, 2021.

5.1.3. Server: Business

Camada responsável por implementar todas as regras de negócio do sistema, com as funções que a aplicação deve fornecer, além de realizar as validações necessárias para o funcionamento do serviço, fornecer e consumir os modelos na visão do negócio e realizar integração com sistemas externos.

Figura 5 - Representação arquitetural da camada Business do servidor

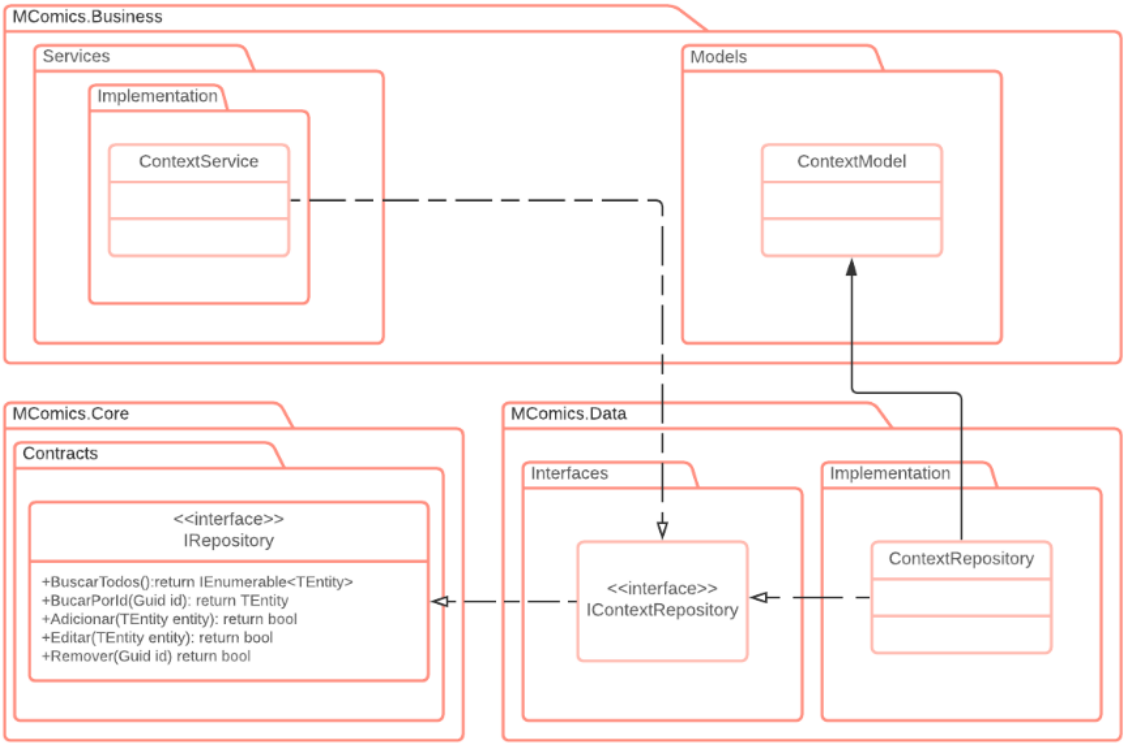


Fonte: Elaborada pelos autores, 2021.

5.1.4. Server: Data

Camada responsável por realizar a integração do serviço com sistemas gerenciadores de banco de dados e garantir a persistência de dados, consumindo os modelos de negócio da camada de business.

Figura 6 - Representação arquitetural da camada Data do servidor



Fonte: Elaborada pelos autores, 2021.

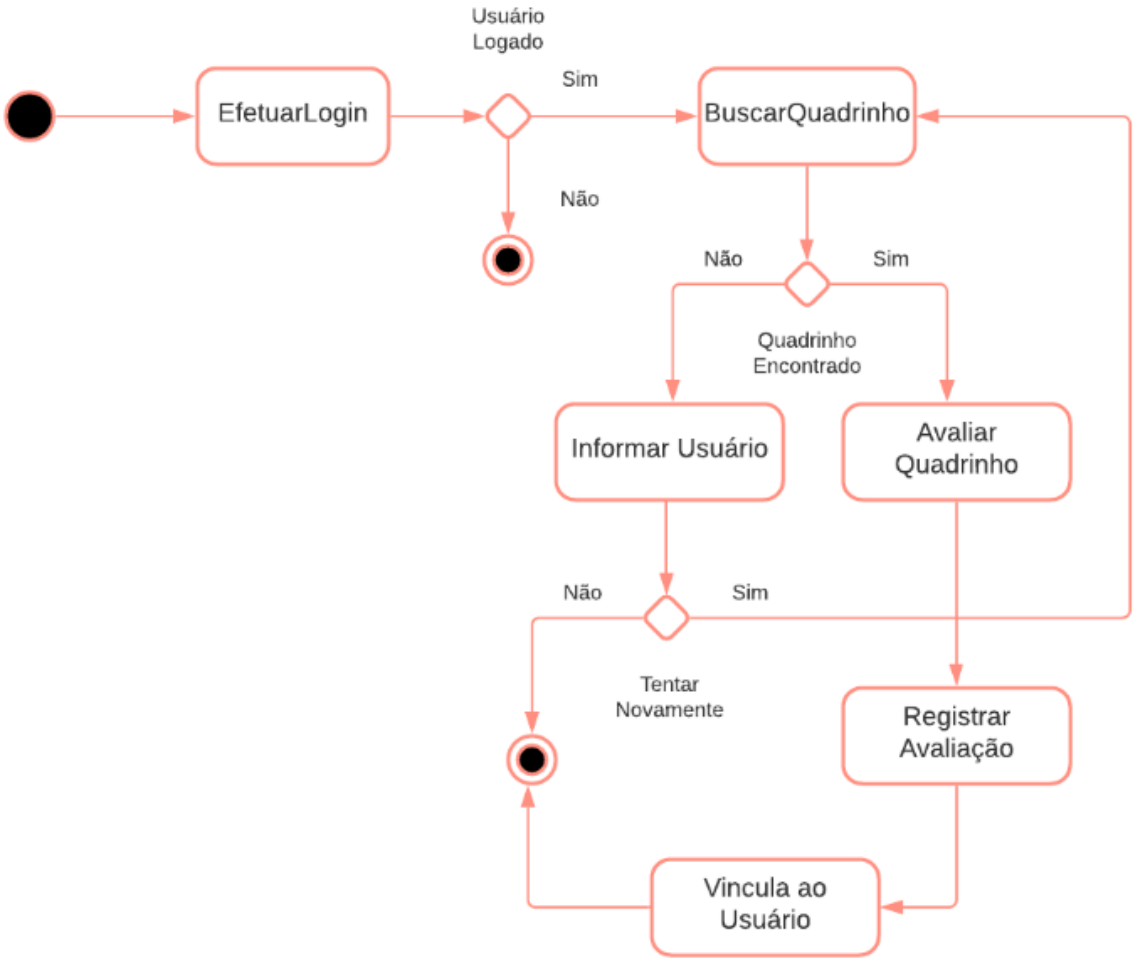
6. Visão de processos

Como apontado na Figura 7, utilizando uma funcionalidade básica do sistema, o caso de uso “Avaliar quadrinho”, é possível compreender o fluxo principal de processos no sistema, para que a função seja executada e suas saídas geradas.

Já na Figura 8, é descrito como esses processos são organizados e como eles se comunicam estando em componentes diferentes. Essa perspectiva é adequada à visão de implantação, visto que o fluxo do processo é executado em seus componentes de hardware.

6.1. Diagrama de processos

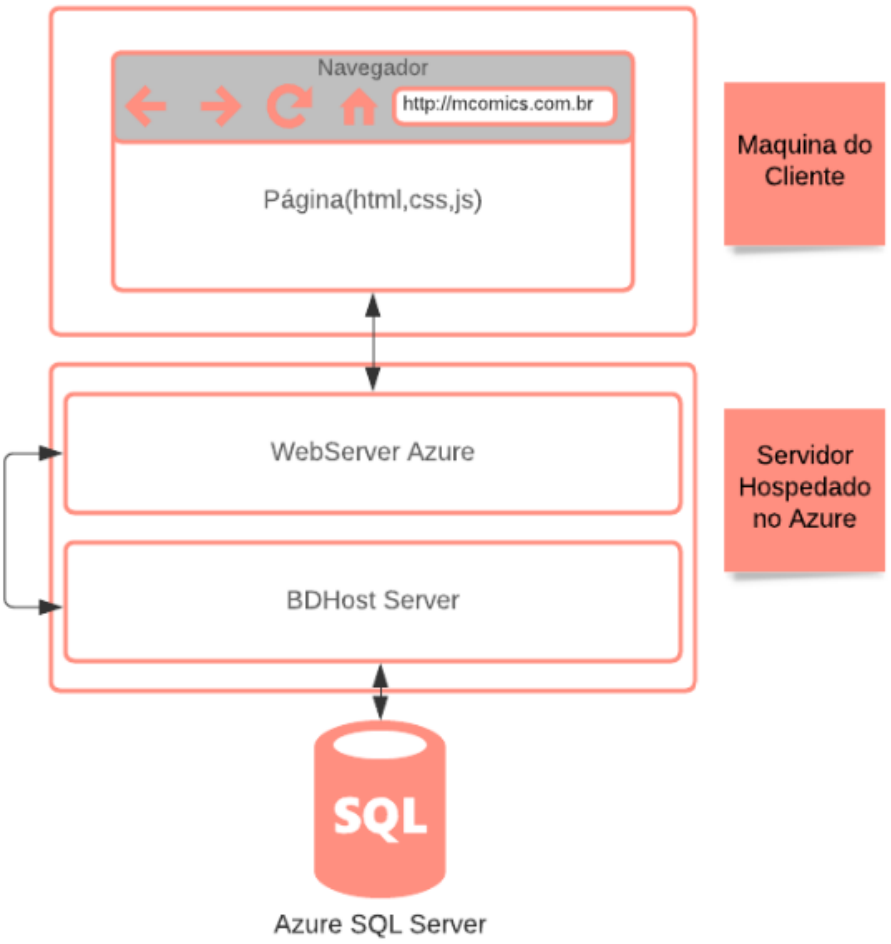
Figura 7 - Diagrama de processos



Fonte: Elaborada pelos autores, 2021.

6.2. Diagrama de fluxo dos processos

Figura 8 - Diagrama de fluxo de processos



Fonte: Elaborada pelos autores, 2021.

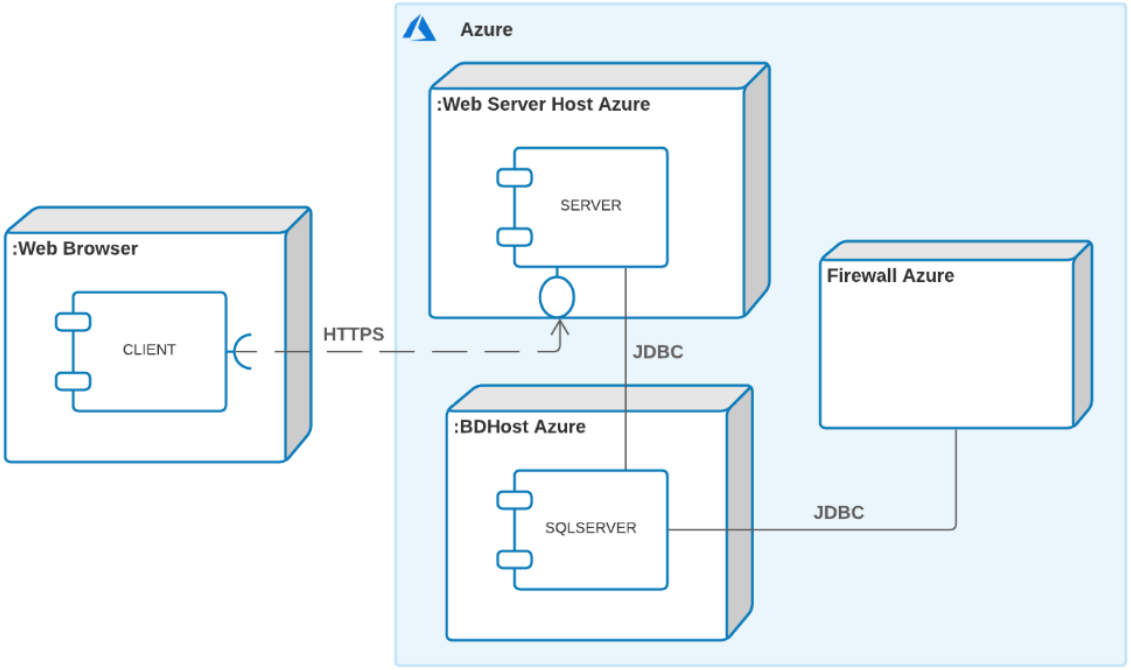
7. Visão de implantação

O diagrama de implantação apontado na subseção abaixo descreve a configuração e distribuição dos principais componentes de hardware utilizados para hospedagem e execução do software.

O “Client” será hospedado e executado no próprio dispositivo do cliente, realizando sua conexão com “Server” que está hospedado em um servidor dentro do Azure. Em contrapartida, o banco de dados da aplicação está localizado em outro servidor responsável apenas pela implantação da persistência, em conjunto com o framework de segurança do Azure.

7.1. Diagrama de implantação

Figura 9 - Diagrama de implantação



Fonte: Elaborada pelos autores, 2021.

8. Visão da implementação

8.1. Visão geral

Como já especificado em tópicos anteriores, o sistema será implementado no estilo arquitetônico de cliente servidor, sendo subdividido em dois módulos o “Cliente” e o “Server”.

8.1.1. Client

O módulo de “Client” será implementado utilizando o estilo arquitetônico monolítico tendo como principal responsabilidade a exibição da interface gráfica para o usuário.

8.1.2. Server

O módulo de “Server” será construído utilizando o estilo arquitetônico 3-Tier Architecture, sendo dividido em três camadas: API, Business e Data.

8.2. Conceitos

O sistema será desenvolvido utilizando os conceitos de Programação Orientada a Objetos, SOLID, DDD, Heurísticas de Nielsen, Clean Code, boas práticas de UX e de Engenharia de Software.

8.3. Frameworks e tecnologias

O sistema utilizará os principais frameworks do mercado para o desenvolvimento de aplicações Web: React, .NET Core 5.0, Entity Framework 5.0, Identity Server 4, Swagger, Azure Host, SQL Server, JWT e HTTPS.

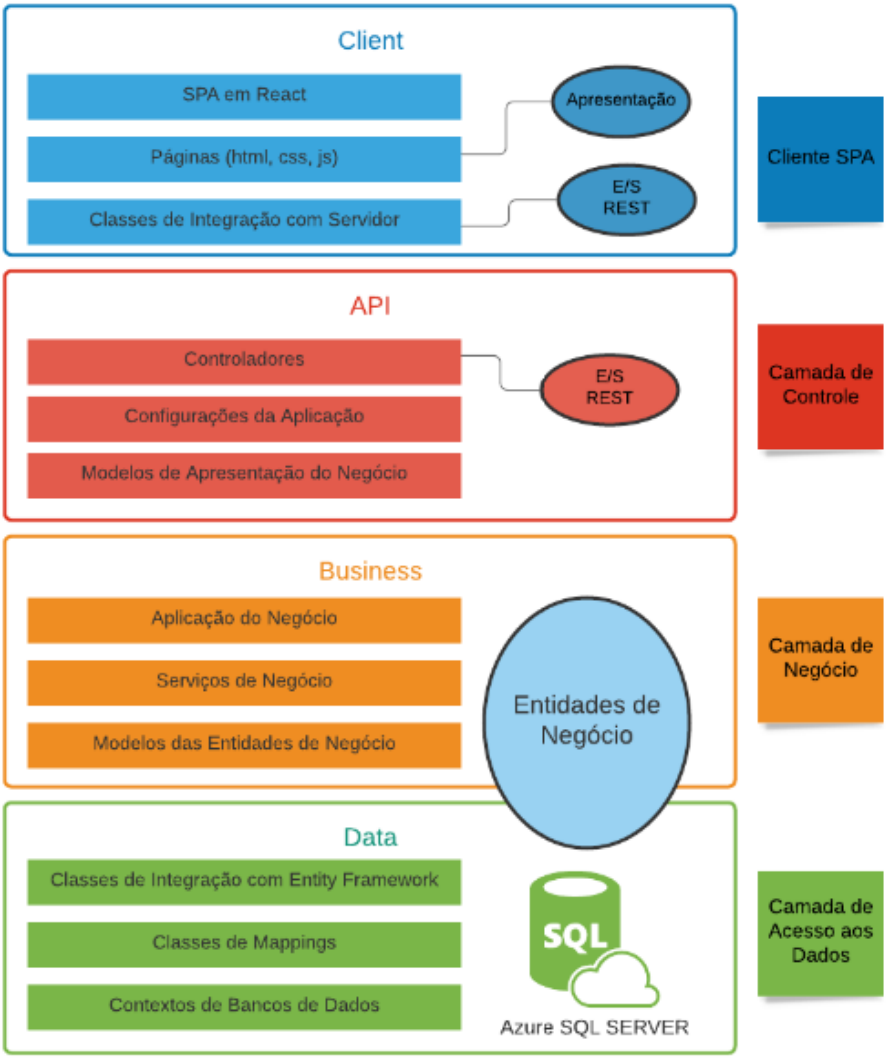
8.4. Linguagens de programação

As linguagens adotadas para o desenvolvimento do sistema foram: TypeScript, C#, SQL, HTML, SCSS.

8.5. Padrões arquiteturais e Padrões de Projeto

Foi adotado para o projeto os seguintes padrões arquitetônicos e padrões de projeto: CQRS, SPA Pattern, Repository Pattern e Generics Pattern.

Figura 10 - Diagrama de implementação



Fonte: Elaborada pelos autores, 2021.

8.6. Camadas

8.6.1. Client

O cliente é uma camada de apresentação que se preocupa em mostrar informações ao usuário e gerenciar toda sua interação. É capaz de receber informações e usar services específicos dos provedores dos servidores. Essa camada utiliza tecnologias de apresentação denominadas páginas de interface rica.

8.6.2. Server

O servidor tem como responsabilidade fornecer todos os dados necessários para a alimentação do “Client” além de atender as requisições apresentadas pelo mesmo. Em consequência disso sua competência se estende a outras funções como

persistência dos dados, validações, segurança, integração com sistemas externos, implementação e funcionamento das regras de negócio.

8.6.2.1. API

Essa camada vem com o objetivo de tratar todas as regras de negócio, que vem através das solicitações do usuário que são tratadas nas rotas HTTPS, manter e consumir os modelos de negócio no formato de apresentação ao usuário e as configurações de todo o “Server”.

8.6.2.2. Business

A camada de negócio encapsula todas as regras de negócio do sistema disponibilizando para a requisição das chamadas em níveis superiores através das injeções de dependências por meio de interface. Consequentemente as aplicações que fornecem todas as funcionalidades do sistema e seus serviços utilizados para a execução de tais aplicações se concentram nesta camada. Por fim, os modelos de entidades na visão de negócio, validação e comunicação com sistemas externos também estão presentes.

8.6.2.3. Data

A camada de dados faz o mapeamento entre o modelo entidade na visão de negócio para sua representação relacional, sendo responsável pelo gerenciamento de todos os CRUDs e consultas, além disso, contém todas as classes necessárias para a integração com servidor de banco de dados.

9. Visão de dados

A versão mínima para a execução deste projeto do .NET Core é a 5.0. Em consequência disto, o React precisa estar atualizado na versão 17.0.2, visto que a integração entre as duas fontes precisam estar sintonizadas, influenciando seriamente na manipulação dos dados enviados pelo protocolo HTTPS.

10. Tamanho e desempenho

Tabela 2 - Requisitos de tamanho e desempenho da arquitetura

Requisito	Descrição
Tempo de resposta	O MComics deverá retornar o resultado no tempo médio de 60 segundos, podendo chegar no máximo a 90 segundos para concluir uma transação.
Capacidade	O software será projetado para suportar 20 usuários utilizando a aplicação simultaneamente, sem perda de velocidade e aumento de tempo de resposta em suas operações.

Fonte: Elaborada pelos autores, 2021.

11. Qualidade

Tabela 3 - Requisitos de qualidade da arquitetura e do software

Requisito	Descrição
Interoperabilidade	Todas as integrações com outros sistemas exteriores deverão acontecer por meio web services utilizando o formato JSON para as mensagens e o protocolo RESTful.
Autenticidade	Toda validação, verificação, autenticação e criação de contas será administrado pelo Identity Server da Microsoft.
Modificabilidade	O sistema deve ser capaz de receber manutenções, atualizações e refatorações de forma eficaz, garantindo a integridade do produto.
Acessibilidade	A interface do usuário deve estar de acordo com as Diretrizes de Acessibilidade para o Conteúdo da Web (WCAG 2.1).

Fonte: Elaborada pelos autores, 2021.

Tabela 4 - Padrões arquiteturais que influenciam a qualidade do software

Padrão arquitetural	Finalidade
CQRS	Considerando a necessidade da aplicação receber futuramente múltiplos usuários, será implementado o padrão arquitetural de baixo nível CQRS, separando as consultas de forma síncrona (Query) e as gravações de forma assíncrona (Commands).
SPA Pattern	A criação da aplicação como uma Single Page Application aumenta o desempenho do software ao carregar recursos HTML, CSS e JavaScript ou TypeScript assim que o site é carregado, fazendo com que o tempo de espera seja o menor possível. Isso reflete diretamente na demanda pelo sistema, além de melhorar a experiência do usuário e a otimização de SEO.
Repository Pattern	Devido à escalabilidade e manutenibilidade da aplicação será implementado o padrão de projeto Repository Pattern, para concentrar as responsabilidades da persistência, evitando duplicidade e influências em outros componentes do negócio.

Generics Pattern	Devido à escalabilidade e manutenibilidade da aplicação será implementado o padrão de projeto Generics Pattern, para fornecer interfaces padrões para os principais componentes do negócio.

Fonte: Elaborada pelos autores, 2021.

12. Referências

[1]ARRUDA, Abigail; FERRAZ, Jacob. **Casos de Uso**. 2021. Disponível em: https://github.com/abigailarruda/DS-ES-2020-2-MComics/blob/f70f2abc6bdbe719536f66cb5bad9b51e0e887e3/docs/01.%20defini%C3%A7%C3%A3o%20de%20requisitos/02.%20casos%20de%20uso.png. Acesso em: 27 abr. 2021.

CLAUDINEY. **Projeto TRIphibius VIGilante - TRIVIG: Documento de Arquitetura de Software.** 2003. Disponível em: https://turing.inf.ufg.br/mod/resource/view.php?id=21191. Acesso em: 26 abr. 2021.

EMPRESA BRASILEIRA DE PESQUISA AGROPECUÁRIA (Brasil). **Documento de Referência para Desenvolvimento de Software da Embrapa**. 2015. Disponível em: https://turing.inf.ufg.br/mod/resource/view.php?id=21192. Acesso em: 26 abr. 2021.

GEERY, Tyler; HICKSON, Maddison; KLIMKOWSKY, Casey; NELSON, Emma. **Software Architecture Documentation**. 2015. Disponível em: http://www.se.rit.edu/~co-operators/SoftwareArchitectureDocumentation.pdf. Acesso em: 28 abr. 2021.

HEITOR; RENÉ, José; LISBOA, Flávio. **Aplicações Demoiselle: Documento de Arquitetura de Referência**. 2008. Disponível em: https://turing.inf.ufg.br/mod/resource/view.php?id=21189. Acesso em: 26 abr. 2021.

REINALDO, Guilherme Alexandre Monteiro. **AMADEUS MM: Documento de Arquitetura**. 2005. Disponível em: https://turing.inf.ufg.br/mod/resource/view.php?id=21190. Acesso em: 26 abr. 2021.

Secretaria da Fazenda do Distrito Federal. **Documento de Arquitetura de Software**. 2016. Disponível em: https://turing.inf.ufg.br/mod/resource/view.php?id=21188. Acesso em: 24 abr. 2021.

SOMMERVILLE, Ian. **Software Engineering**. 10. ed. Massachusetts: Pearson, 2016.

What is three-tier architecture? 2021. Disponível em: https://www.ibm.com/cloud/learn/three-tier-architecture. Acesso em: 28 abr. 2021.