1. Introdução

1.1 Finalidade

Este documento possui como objetivo definir os aspectos da Arquitetura do software e é direcionado aos stakeholders do software a ser desenvolvido, tais como: Gerentes do Projeto, Clientes e equipe técnica, possuindo grande foco para os Desenvolvedores e a Equipe de implantação.

1.2 Escopo

Este documento se baseia no documento de requisitos do Projeto de Transferência do Cuidado de Pacientes para definir os atributos de qualidade a serem priorizados, bem como, os estilos arquiteturais que favorecem tais atributos e as representações das visões arquiteturais e seus sub-produtos.

1.3 Definições, Acrônimos e Abreviações

AAS: Artefato de Arquitetura de Software.
RAS: Requisito de Arquitetura de Software.
Id.: Identificador.
Software: Conjunto de documentações, guias, metodologias, processos, códigos e ferramentas para a solução de um problema.
Sistema: Conjunto de pessoas, softwares, hardwares e outros sistemas para a solução de um problema.
Stakeholder: Indivíduo, grupo ou organização que possua interesse no Sistema.
Visão Arquitetural: Produto resultante da interpretação de um Stakeholder do sistema.
Ponto de Vista Arquitetural: Produto resultante da execução de uma Visão Arquitetural.
Arquitetura de Software: Forma como os componentes são agrupados com o objetivo de construir um software ou sistema.
Trade-Off: Cada arquitetura de software possui seus atributos de qualidade que são favorecidos e desfavorecidos, o trade-off consiste em ter a consciência dessas características para escolher uma arquitetura que favorece os atributos de qualidade priorizados.
Atributos de qualidade: São atributos que impactam diretamente na concepção de um software, são definidos conforme a ISO-IEEE 9126.
Sistema Operacional: Software responsável por gerenciar e abstrair a interação entre o usuário e o hardware ou aplicações externas e o hardware.
UML: Sigla para Linguagem de Modelagem Unificada.
HTTP: Sigla para Protocolo de Transferência de Hipertexto.
Nó-físico: Termo para representar um componente físico de modo geral, como um navegador ou um banco de dados, por exemplo.

1.4 Referências

Id.	Nome do artefato
AAS_1	Transferência do Cuidado de Pacientes 3.0 - Documento de Requisitos
AAS_2	ISO-IEEE 9126
AAS_3	ISO-IEEE 42010
AAS_4	Slides Ministrados em Sala
AAS_5	4+1 View

1.5 Visão Geral

Os próximos tópicos descrevem quais serão os requisitos e restrições utilizados para definir a arquitetura a ser implementada, bem como, quais atributos de qualidades serão priorizados e o porquê da escolha. Quais os padrões arquiteturais serão utilizados conforme os atributos de qualidade selecionados e como funcionará o trade-off entre esses padrões arquiteturais, bem como o porquê da escolha dos padrões arquiteturais. Quais e como as visões arquiteturais serão detalhadas e quais os pontos de vista da arquitetura serão utilizados para descrever as visões.

2. Contexto da Arquitetura

2.1 Funcionalidades e Restrições Arquiteturais

O CQRS fornece uma abordagem arquitetural que separa as responsabilidades de leitura e escrita de forma a melhorar a escalabilidade e a performance da aplicação. No entanto, a adoção do CQRS exige a adesão a algumas restrições arquiteturais para o bem-estar da arquitetura. Por exemplo, a arquitetura CQRS exige que as solicitações de leitura e escrita sejam tratadas de forma independente, e que eles não se misturem. Além disso, o CQRS define que os estados de dados não podem ser alterados diretamente através de fontes externas, como serviços web ou APIs, e que tais alterações de estado devem ser feitas apenas através do uso de comandos. Finalmente, a arquitetura CQRS exige que os

> dados de leitura sejam armazenados de forma separada dos dados de escrita.(Ahmed et al., 2017).

Outra restrição arquitetural do CQRS é o uso de um modelo de dados orientado a comando para representar os dados de escrita. Portanto, para aplicar o CQRS, os desenvolvedores devem desenvolver modelos de dados orientados a comandos que permitam que os comandos sejam convertidos efetivamente em operações de escrita. Além disso, a arquitetura CQRS exige que os comandos sejam tratados de forma assíncrona. Isto significa que os comandos são armazenados em uma fila de comandos, que é processada de forma assíncrona, permitindo que os comandos sejam executados em um contexto de thread separado.

Por último, o CQRS exige que os dados de leitura e escrita sejam armazenados em fontes de dados independentes. A arquitetura CQRS não exige que os dados de leitura e escrita sejam armazenados em bancos de dados separados, mas sim que eles sejam armazenados em fontes de dados separadas.

2.2 Atributos de Qualidade Prioritários

Atributos de qualidade prioritários, como **segurança** e **usabilidade**, são fundamentais para uma arquitetura de software bem sucedida. De acordo com o documento de estilo de arquitetura de software da IEEE, "os principais atributos de qualidade devem ser considerados ao projetar a arquitetura de software, e podem incluir segurança, usabilidade, escalabilidade, desempenho, fiabilidade, extensibilidade, flexibilidade e manutenibilidade" (IEEE, 2005, p. 4).

Em particular, a segurança é um dos principais atributos de qualidade de um sistema de software, pois garante que os dados e recursos sejam acessados somente por usuários autorizados. Além disso, as arquiteturas de software seguras devem ser projetadas para resistir a ataques maliciosos, garantindo que os usuários não possam acessar dados confidenciais ou modificar recursos não autorizados (Phillips e Rabhi, 2016).

Por outro lado, a usabilidade também é um aspecto importante na arquitetura de software e microsserviços. Esta característica se refere à facilidade de uso do sistema por parte dos usuários, garantindo que os usuários possam acessar os recursos de maneira rápida e eficiente. As arquiteturas de software usáveis devem incluir interfaces intuitivas e fáceis de usar, além de permitir aos usuários acessar os recursos de maneira rápida e eficiente (Phillips e Rabhi, 2016).

Ao escolher segurança e usabilidade como atributos de qualidade prioritários na arquitetura de microsserviços, existem alguns trade-offs a serem considerados. Em particular, a segurança é fundamental para garantir que os dados e recursos sejam acessados somente por usuários autorizados, mas isso pode resultar em um desempenho reduzido se a segurança for exagerada. Além disso, a segurança pode aumentar o custo de desenvolvimento, pois implica um esforço maior na implementação de medidas de segurança (Phillips e Rabhi, 2016). Já a usabilidade é importante para garantir que os

usuários possam acessar os recursos de maneira rápida e eficiente, mas em alguns casos pode ser sacrificada em favor de outros atributos de qualidade como segurança, escalabilidade e desempenho (Phillips e Rabhi, 2016).

Assim, segurança e usabilidade são atributos de qualidade prioritários essenciais para as arquiteturas de software bem-sucedidas, pois garantem que os dados e recursos sejam acessados somente por usuários autorizados e que os usuários possam acessar os recursos de maneira rápida e eficiente.

3. Representação da Arquitetura

O CQRS (Segregação de Responsabilidade de Comando e Consulta), é um padrão de arquitetura que separa as operações de leitura e atualização de um banco de dados. A implementação do CQRS em seu aplicativo pode maximizar o desempenho, a escalabilidade e a **segurança**. A flexibilidade criada pela migração para CQRS permite ao sistema evoluir melhor ao longo do tempo e impede que os comandos de atualização causem conflitos de mesclagem no nível de domínio.

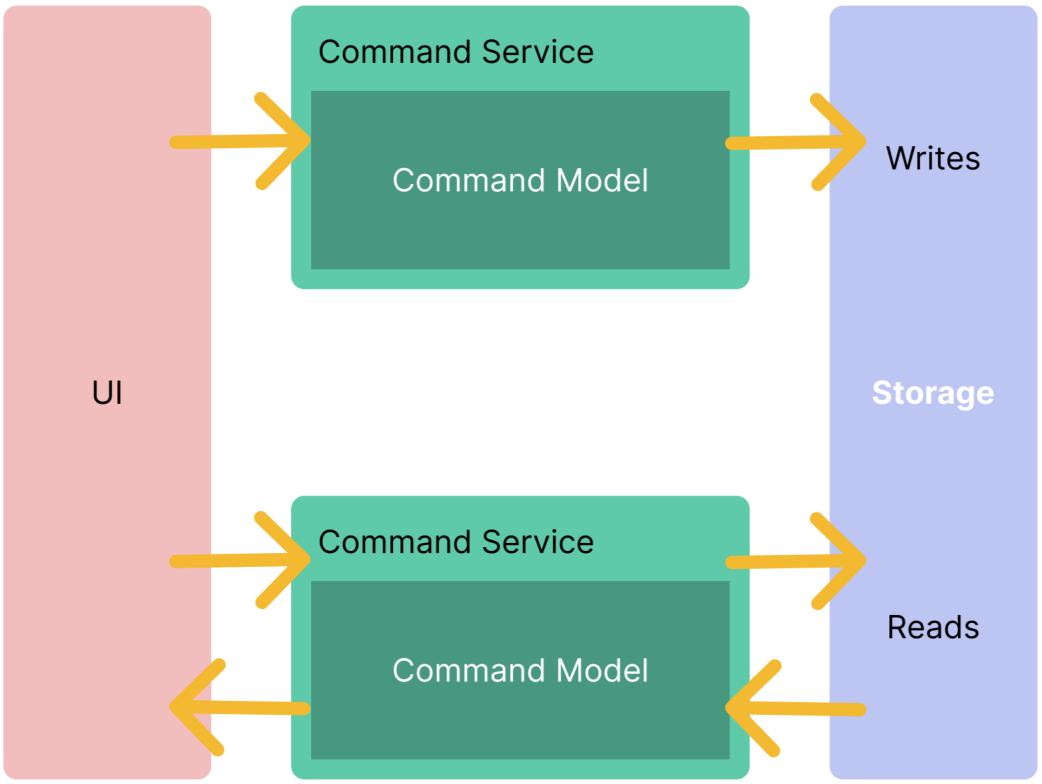
O uso da arquitetura CQRS (Segregação de Responsabilidade de Comando e Consulta) tem se tornado cada vez mais comum na área de desenvolvimento de software. Esta abordagem é baseada na ideia de que os comandos e as consultas devem ser tratados de forma separada, permitindo que as responsabilidades de cada um sejam claramente definidas. Como afirmou Martin Fowler: 'CQRS é uma arquitetura que separa os comandos que mudam o estado de um sistema de suas consultas para os dados desse sistema'. Esta separação permite que o código seja mais simples, o que resulta em melhor desempenho e escalabilidade. Além disso, como afirmou Greg Young, 'CQRS nos dá a capacidade de nos concentrarmos em cada camada de forma independente, permitindo que cada camada seja otimizada para sua própria responsabilidade'. Portanto, CQRS é uma abordagem poderosa para desenvolver aplicativos modernos e escaláveis.

O CQRS separa as leituras e gravações em modelos separados, usando comandos para atualizar dados e consultas para ler dados.

- Os comandos devem ser baseados em tarefas, em vez de centrados nos dados. ("Book hotel room", não "set ReservationStatus to Reserved").
- Os comandos podem ser colocados em uma fila para processamento assíncrono, em vez de serem processados de forma síncrona.
- As consultas nunca modificam o banco de dados. Uma consulta retorna um DTO que não encapsula qualquer conhecimento de domínio.

Segue abaixo uma ilustração representando o padrão arquitetural CQRS

Imagem 1 - Representação do padrão arquitetural CQRS



fonte: Elaborado pelos autores.

4. Ponto de vista dos Casos de Uso

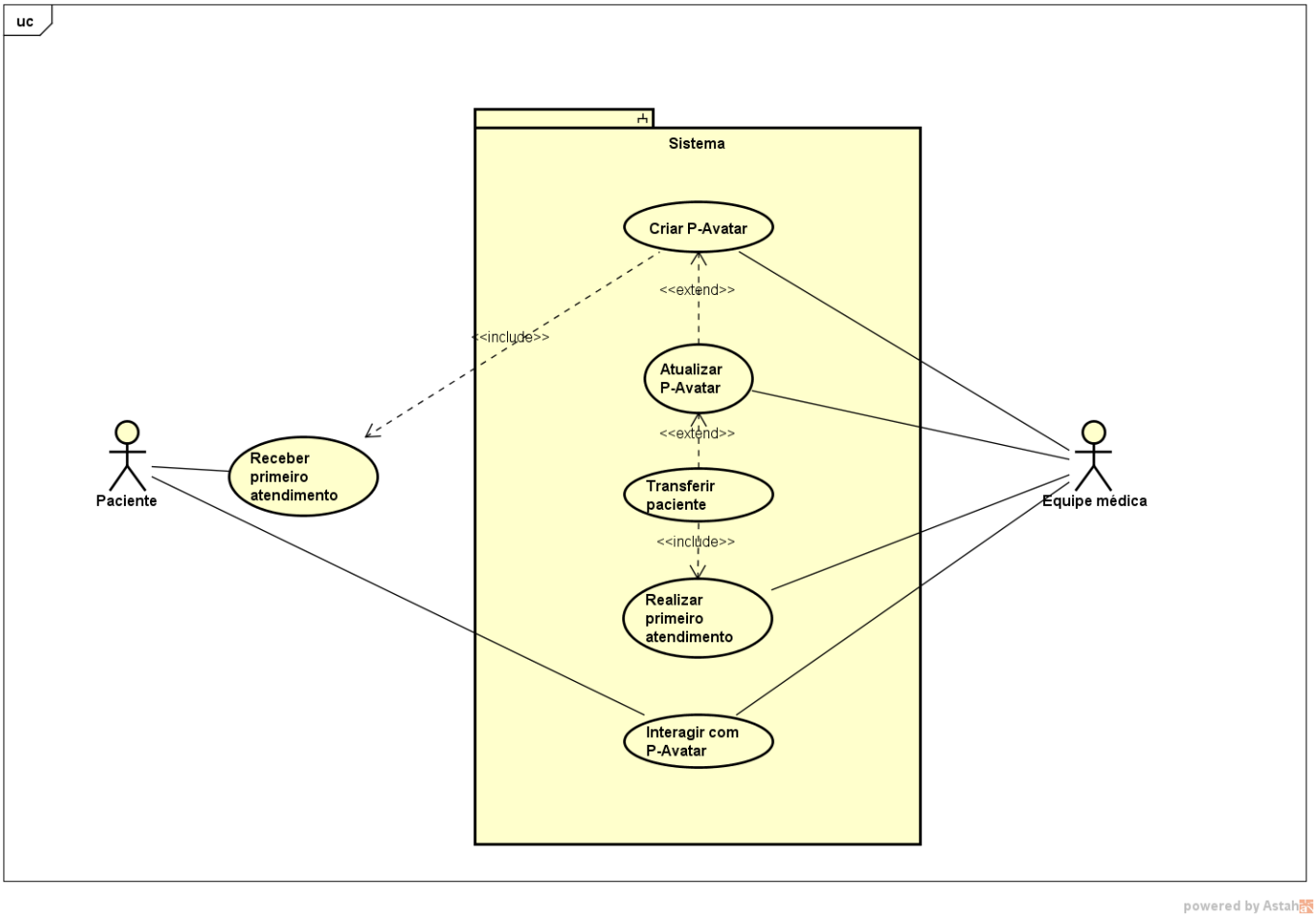
4.1 Descrição

A visão de de casos de uso é uma visão com a finalidade de fornecer fornecer uma base para o planejamento da arquitetura e de todos os outros artefatos que serão gerados durante o ciclo de vida do software. Ela ilustra os casos de uso e cenários que englobam o comportamento, as classes e riscos técnicos significativos do ponto de vista da arquitetura.

4.2 Visão de Casos de Uso

Cada requisito funcional definido em AAS_1 foi considerado um caso de uso e analisado de forma a gerar o diagrama de casos de uso do software a ser desenvolvido.

Imagem 2 - Diagrama de Caso de Uso



fonte: Elaborado pelos autores.

5. Ponto de vista do Projetista

5.1 Visão Geral

O ponto de vista do projetista é direcionado aos projetistas e desenvolvedores do software e tem como objetivo definir as principais partes que o compõem, tal como os componentes, além de definir quais as suas responsabilidades. Foi escolhida por ser uma visão primordial para a compreensão do software e de todo o seu ecossistema. O modelo arquitetural proposto para a construção deste software será composto por 4 (quatro) componentes essenciais: Paciente, Equipe Médica, Atendimento e Avatar.

5.2 Visão de Componentes

Visão: A visão deste sistema é a de proporcionar uma experiência de atendimento de qualidade ao paciente. A equipe médica tem a oportunidade de ver o paciente em um ambiente virtual e acompanhar as suas modificações durante o atendimento.

Modelo Conceitual: O modelo conceitual do sistema compreende as entidades Paciente, Equipe Médica, Atendimento e Avatar. O Atendimento conecta o Paciente à Equipe Médica. O Avatar é gerado durante o Atendimento e reflete as modificações que ocorrem no Paciente.

Modelo de Dados: O modelo de dados do sistema armazena os dados de cada entidade (Paciente, Equipe Médica, Atendimento e Avatar) e as informações sobre as suas relações. Modelo de Processos: O sistema conta com dois processos principais: o processo de Atendimento e o processo de Geração do Avatar. O processo de Atendimento é iniciado quando o Paciente é conectado à Equipe Médica. O processo de Geração do Avatar é iniciado quando o Atendimento é iniciado e é responsável por gerar e manter o Avatar virtual.

Comandos: Os Comandos representam as operações que manipulam os dados do sistema. Neste caso, os Comandos são usados para iniciar o Atendimento, criar o Avatar e registrar as modificações no Paciente.

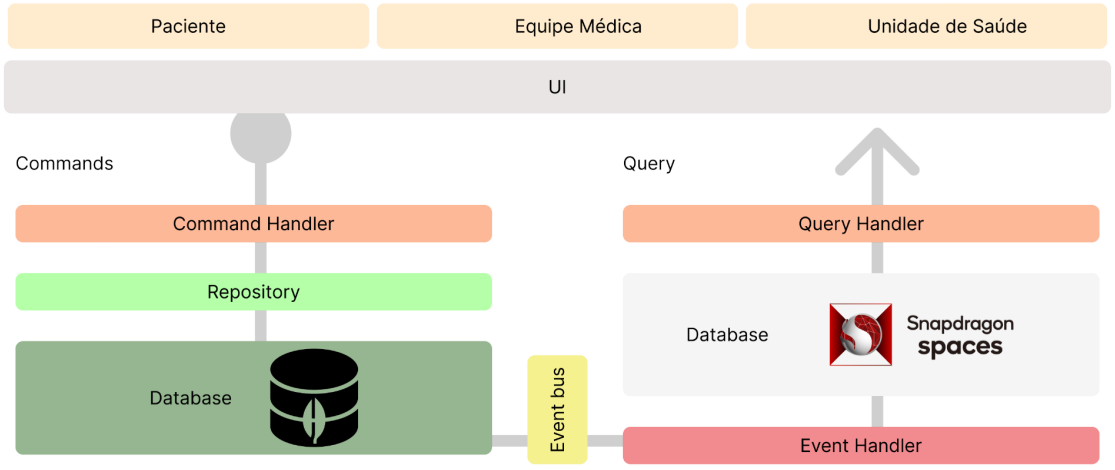
Querys: As Querys são usadas para recuperar os dados do sistema. Neste caso, as Querys são usadas para recuperar os dados do Paciente, da Equipe Médica, do Atendimento e do Avatar.

5.3 Detalhamento das Camadas

Os Comandos são responsáveis por escrever as chamadas para o banco de dados a partir da ação do usuário dentro do ambiente virtual que afetará o Avatar. Esse componente é acessado a partir das entidades Paciente e/ou Equipe médica, como exemplificado no diagrama de Caso de Uso, responsável pelas requisições POST.

As Querys são responsáveis por ler as modificações realizadas dentro das tabelas no banco de dados e atualizar na UI essas alterações para o usuário. Esse componente é representado como Sistema dentro do diagrama de Caso de Uso, responsável pelas requisições GET.

Imagem 3 - Design da Arquitetura



fonte: Elaborado pelos autores.

6. Ponto de vista do Desenvolvedor

6.1 Visão Geral

O ponto de vista do desenvolvedor é direcionado aos projetistas e desenvolvedores do software e tem como objetivo definir as principais partes responsáveis por definir as funcionalidades e restrições do software, tal como as classes.

6.2 Visão lógica

A visão lógica é responsável por definir como a estrutura dos componentes do software será realizada e foi escolhida para auxiliar na construção da arquitetura proposta.

6.2.1 Detalhamento das classes

Primeiramente temos três classes que representam entidades reais: Paciente, EquipeSaude e UnidadeAtendimento. A classe Paciente será responsável por representar o paciente no sistema, os seus atributos são: id, tipoAtendimento, prontuario, situacao, acao. O atributo prontuário provém de uma classe à parte nomeada Prontuário, segundo o princípio KISS(Keep it simple, stupid!).

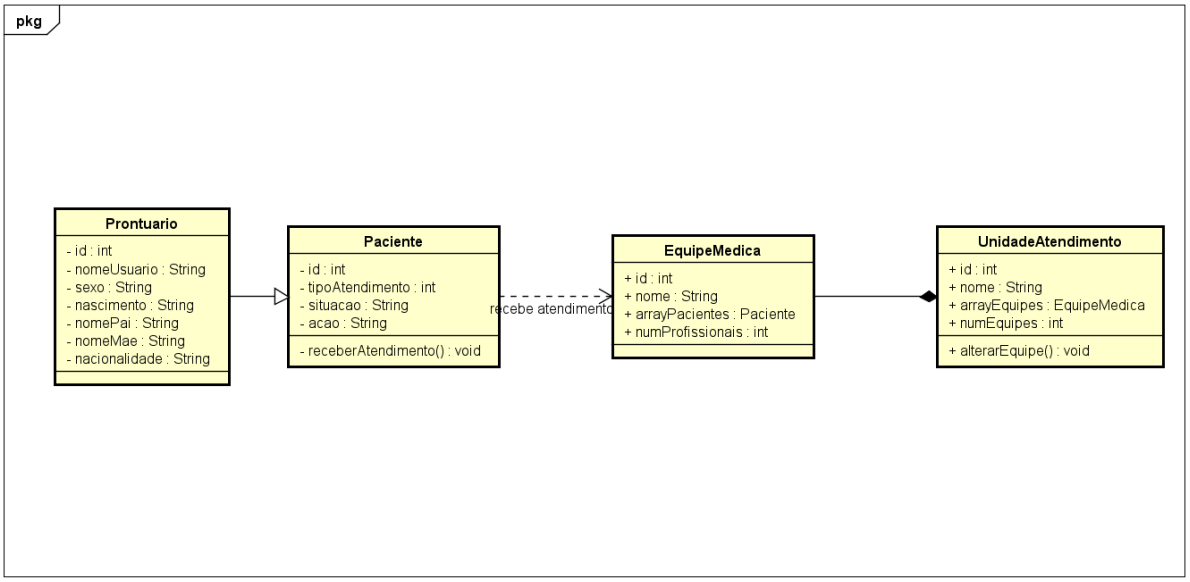
A classe EquipeSaude é a responsável por representar as equipes de saúde e os seus atributos, os quais são: id, nome, arrayPacientes, numProfissonais. A classe UnidadeAtendimento é a classe para representar no sistema as unidades de atendimento, possuindo os seguintes atributos: id, nome, arrayEquipes, numEquipes. Outra classe

presente no sistema é Avatar, que será o gêmeo digital do paciente, ou seja, possui as mesmas características da classe Paciente.

Para o acesso ao banco de dados teremos duas classes com funções distintas, Leitor e Escritor. O primeiro será responsável por fazer as operações de leituras no banco de dados e o último terá a responsabilidade de fazer as operações de escrita.

As classes descritas podem ser visualizadas no Diagrama de classes UML abaixo:

Imagem 4 - Diagrama de Classes



fonte: Elaborado pelos autores.

6.3 Visão de segurança

As camadas Command Handler e Query Handler são, respectivamente, camadas de segurança para fazer chamadas de escrita e leitura dos bancos de dados, tratando as chamadas HTTP de forma segura a fim de garantir uma camada de segurança eficaz. Os dados modificados são então atualizados dentro do DB do snapdragon database que são traduzidos em modificações dentro do ambiente virtual.

6.3.1 Detalhamento da segurança

As tratativas e testes unitários dentro dos componentes Command e Query serão feitas utilizando as bibliotecas disponíveis em Java, como o JUnit, a fim de cobrir uma boa porcentagem de código testada. O JUnit pode oferecer uma forma eficaz de testar a segurança de aplicações com arquitetura CQRS. Utilizando ferramentas de teste de unidade, é possível testar rapidamente se os componentes da aplicação estão funcionando de maneira segura. Por meio de ferramentas como JUnit, é possível executar testes

de unidade para garantir que as regras de segurança sejam seguidas e as senhas e outras informações confidenciais não sejam comprometidas.

Além disso, utilizando o JUnit, é possível testar as habilidades do sistema para detectar e bloquear ataques de hackers. Por meio de arquitetura CQRS, o JUnit pode ser usado para testar os recursos e os mecanismos de segurança usados para aplicações em tempo real.

7. Ponto de vista do Implantador

7.1 Visão Geral

A visão de implantação é direcionada para a equipe de implantação e é responsável por definir as ferramentas e ambiente necessário para o bom funcionamento do software. Ela foi escolhida por se tratar de um software com múltiplos componentes independentes e a necessidade de coexistir com um ecossistema potencialmente mutável.

7.2 Visão Lógica

As ferramentas utilizadas para implementação da arquitetura serão:

Visual Studio code Version: 1.74.3 (user setup)
Commit: 97dec172d3256f8ca4bfb2143f3f76b503ca0534
Date: 2023-01-09T16:59:02.252Z
Electron: 19.1.8
Chromium: 102.0.5005.167
Node.js: 16.14.2
V8: 10.2.154.15-electron.0
OS: Windows_NT x64 10.0.22621
Sandboxed: No

Snapdragon_Spaces_SDK_for_Unreal_0_9_0, disponível em: https://spaces.qualcomm.com/sdk/

Unreal Engine 5.1.0

Referências

IEEE. (2005). IEEE Standard for Software Architecture Description. IEEE Standard 1471-2000. Recuperado de https://ieeexplore.ieee.org/document/1145796

Phillips, T. E. Rabhi, F. (2016). Projeto de software: Teoria e prática. 5ª ed. Rio de Janeiro: Elsevier.

Fowler, M. (2011). CQRS. Consultado em 9 de dezembro de 2022, de https://martinfowler.com/bliki/CQRS.html

Young, G. (2008). CQRS, Task Based UIs, Event Sourcing agh! Consultado em 9 de dezembro de 2022, de https://codebetter.com/gregyoung/2008/04/09/cqrs-task-based-uis-event-sourcing-agh/

Microsoft. Padrão CQRS. Consultado em 9 de dezembro de 2022, https://learn.microsoft.com/pt-br/azure/architecture/patterns/cqrs

Pandey, D., & Singh, A. (2019). Unit Testing for Performance Improvement of CQRS Applications. International Journal of Emerging Trends & Technology in Computer Science, 8(2), 28-34.

Ozakin, T., & Kocoglu, B. (2015). Testing Aspects of the CQRS Pattern. International Journal of Computer Applications, 119(20), 1-5.

Ahmed, M. S., Salem, M., Hossain, M. A., & Mozumder, S. (2017). Um estudo sobre arquitetura CQRS. In International Conference on Solutions for Science, Engineering and Technology (pp. 627-637). Springer, Cham.