

Faculdade

**XPe**



# RELATÓRIO

---

PROJETO  
APLICADO

---

PÓS-GRADUAÇÃO

**XP Educação**  
**Relatório do Projeto Aplicado**

# Arquitetura de Referência para um Sistema de Reservas de Hotel

Wesley Lucas da Costa

Orientador(a): Ricardo Brito Alves

Julho de 2024



Wesley Lucas da Costa

XP EDUCAÇÃO

RELATÓRIO DO PROJETO APLICADO

# Arquitetura de Referência para um Sistema de Reservas de Hotel

Relatório de Projeto Aplicado desenvolvido para fins de conclusão do curso Arquitetura de Software e Soluções.

Orientador (a): Ricardo Brito Alves

Campos do Jordão

Julho de 2024

# Sumário

1.1	Desafio	4
1.1.1	Análise de Contexto	4
1.1.3	Benefícios e Justificativas	9
1.1.4	Hipóteses	13
1.2	Solução	15
1.2.1	Objetivo SMART	15
1.2.2	Premissas e Restrições	15
1.2.3	Backlog de Produto	18
2.	Área de Experimentação	19
2.1	Sprint 1	19
2.1.1	Solução	19
	• Evidência do planejamento:	19
	• Evidência da execução de cada requisito:	20
	• Evidência dos resultados:	24
2.1.2	Lições Aprendidas	28
2.2	Sprint 2	29
2.2.1	Solução	29
	• Evidência do planejamento:	29
	• Evidência da execução de cada requisito:	30
	• Evidência dos resultados:	34
2.2.2	Lições Aprendidas	37
2.3	Sprint 3	38
2.3.1	Solução	38
	• Evidência do planejamento:	38
	• Evidência da execução de cada requisito:	39
	• Evidência dos resultados:	41
2.3.2	Lições Aprendidas	43
3.	Considerações Finais	44
3.1	Resultados	44
3.2	Contribuições	45
3.3	Próximos passos	46

## 1.1 Desafio

### 1.1.1 Análise de Contexto

A indústria hoteleira enfrenta desafios significativos no gerenciamento de reservas, disponibilidade de quartos e comunicação com os clientes. A crescente demanda por soluções mais ágeis e integradas levou ao desenvolvimento de um sistema de reservas moderno que utiliza microserviços para garantir escalabilidade e flexibilidade.

O principal desafio é criar uma arquitetura de referência para um sistema de reservas de hotel que integre diversas funcionalidades, como gerenciamento de quartos, processamento de reservas e envio de notificações, em uma estrutura robusta e escalável. Além disso, é essencial que o sistema ofereça uma experiência de usuário fluida e eficiente através de uma interface moderna desenvolvida utilizando um SPA.

#### Causas do Problema

- **Sistemas Legados:** Muitos hotéis ainda utilizam sistemas legados que são difíceis de manter e integrar com novas tecnologias.
- **Ineficácia de Comunicação:** A falta de integração entre diferentes serviços pode resultar em comunicação ineficaz com os clientes.
- **Escalabilidade Limitada:** Sistemas tradicionais podem enfrentar dificuldades para escalar com o aumento da demanda.
- **Experiência de Usuário:** Interfaces desatualizadas podem resultar em uma experiência de usuário ruim.

#### Percepções e Informações Utilizadas

- **Pesquisa de Mercado:** Para identificar as principais necessidades e expectativas dos clientes de hotéis.
- **Estudos de Caso de Outras Indústrias:** Para aplicar melhores práticas de sistemas de microserviços e arquitetura orientada a eventos.

## Matriz CSD

A Matriz CSD é uma ferramenta crucial para identificar certezas, suposições e dúvidas cruciais relacionadas aos diferentes atores envolvidos no desenvolvimento do sistema de reservas de hotel. Organizada por atores específicos, esta matriz permite uma análise estruturada que facilita a gestão de riscos e decisões ao longo do projeto.

### Matriz CSD - Gerentes de hotel e viajantes

		CERTEZAS	SUPOSIÇÕES	DÚVIDAS
DIFERENTES ÓTICAS DE ANÁLISE	ATORES	Gerentes de hotel e viajantes frequentes usarão o sistema.	Gerentes têm conhecimento básico de tecnologia.	Como gerentes reagirão à transição dos sistemas legados?
	CENÁRIOS	Gerenciamento de reservas e notificações serão funcionalidades centrais.	O sistema deve escalar para suportar picos de demanda.	Como garantir comunicação eficiente entre microsistemas?
	REGRAS	Reservas devem ser confirmadas por e-mail.	A interface deve ser fácil de usar e autoexplicativa.	Precisa-se de um plano de treinamento detalhado para gerentes?

### Matriz CSD - Desenvolvedores

		CERTEZAS	SUPOSIÇÕES	DÚVIDAS
DIFERENTES ÓTICAS DE ANÁLISE	ATORES	Desenvolvedores envolvidos no projeto.	Viajantes preferem realizar reservas online.	Quais são as expectativas específicas dos viajantes sobre a interface?
	CENÁRIOS	A interface será desenvolvida em Angular e o backend em .NET.	Integração com sistemas de terceiros será necessária.	Quais serão os custos associados à escalabilidade do sistema?
	REGRAS	Microserviços devem ser independentes e escaláveis.	O sistema deve suportar múltiplos métodos de pagamento.	Que tipo de feedback será necessário coletar para melhorar a interface?

## POEMS

O POEMS analisa o contexto do sistema de reservas de hotel, identificando pessoas, objetos, ambiente, mensagens e serviços envolvidos. Esta análise destaca os elementos principais e fornece insights para melhorar a eficiência e a experiência do usuário.

Pessoas	Objetos	Ambiente	Mensagem	Serviços
Quem está presente no contexto em análise?	Que objetos fazem parte do ambiente?	Quais são as características do ambiente?	Que mensagens são comunicadas?	Quais os serviços oferecidos?
Gerentes de hotel	Computadores, tablets, smartphones	Ambiente hoteleiro, recepção, escritório	Confirmação de reservas	Gerenciamento de reservas, alocação de quartos
Viajantes frequentes	Site de reservas	Ambiente online, acessível 24/7	Confirmação de reservas, lembretes de estadia	Reservas online, check-in/check-out
Desenvolvedores back-end	Servidores, bancos de dados	Ambiente de desenvolvimento e produção	Logs de sistema, alertas de performance	Desenvolvimento de APIs, integração de sistemas, manutenção de servidores
Desenvolvedores front-end	Ferramentas de desenvolvimento (IDE, repositórios de código)	Ambiente de desenvolvimento colaborativo	Atualizações de progresso, feedback de usuários	Desenvolvimento de interface, testes de usabilidade
Registros		Insights		
Dados de ocupação, feedback dos clientes, relatórios de performance		<ul style="list-style-type: none"> <li>Necessidade de tecnologias modernas para eficiência, importância de uma boa comunicação</li> <li>Preferência por interfaces intuitivas e rápidas</li> <li>Necessidade de um processo de check-in/check-out eficiente, importância do atendimento ao cliente</li> </ul>		

### 1.1.2 Personas

A criação de personas nos permite compreender as necessidades, comportamentos, objetivos e desafios dos usuários. Identificamos duas personas principais com base em pesquisas: João Silva, gerente de hotel, e Maria Oliveira, viajante frequente. Elas guiam as decisões de design e desenvolvimento da solução.

#### JOÃO SILVA - GERENTE DE HOTEL

João Silva, 45 anos, é um gerente de hotel preocupado com a eficiência operacional e a satisfação do cliente. Ele está constantemente em busca de melhorias tecnológicas para integrar novas soluções que mantenham o hotel competitivo no mercado. João valoriza sistemas que facilitam a gestão de reservas e melhoram a experiência do cliente, buscando sempre eficiência e modernização.





### MARIA OLIVEIRA - VIAJANTE FREQUENTE

Maria Oliveira, 30 anos, é uma viajante frequente que prioriza conveniência e rapidez na reserva de hotéis. Ela prefere sistemas intuitivos e rápidos, valorizando uma interface amigável que ofereça respostas rápidas às suas necessidades. Maria busca uma plataforma de reservas confiável que facilite suas viagens sem complicações, valorizando a eficiência e a praticidade.



### 1.1.3 Benefícios e Justificativas

A seguir, destacamos os principais elementos e benefícios da arquitetura de referência para um sistema de reservas de hotel, visando otimizar eficiência, experiência do usuário, escalabilidade e redução de custos.

#### Benefícios do Projeto

- **Melhoria na Eficiência Operacional:** A arquitetura de microserviços permite uma manutenção mais fácil e uma melhor integração entre diferentes serviços.
- **Melhoria na Experiência do Usuário:** Uma interface moderna em Angular oferece uma experiência de usuário superior.
- **Escalabilidade:** A arquitetura escalável pode lidar com um aumento no volume de reservas sem comprometer o desempenho.
- **Comunicação Eficiente:** O microserviço de notificações garante que os clientes sejam informados em tempo real sobre suas reservas.
- **Redução de Custos:** A automação e a integração eficiente podem reduzir custos operacionais e de manutenção.

#### Justificativas

O investimento na arquitetura de referência para o sistema de reservas de hotel é justificado pelos benefícios esperados, como a melhoria significativa na eficiência operacional e na experiência do usuário. Isso pode resultar em maior satisfação e fidelização dos clientes. Além disso, a escalabilidade do sistema permite que o hotel lide com picos de demanda sem comprometer a qualidade do serviço, gerando novas oportunidades de receita.

## Blueprint

O blueprint das principais funcionalidades do sistema de reservas de hotel destaca a eficiência operacional e a satisfação do cliente como objetivos centrais. Através de uma arquitetura de referência baseada em microserviços, o sistema oferece uma interface moderna e intuitiva, comunicação em tempo real e atualização precisa de disponibilidade de quartos. As atividades e processos de suporte são cuidadosamente delineados para garantir que cada funcionalidade opere de maneira eficiente, proporcionando uma experiência superior tanto para os clientes quanto para os desenvolvedores.

**Tabela 01: Funcionalidade de Gerenciamento de Reservas**

Funcionalidade	Detalhes
Ações do Cliente	Realizar, modificar e cancelar reservas de quarto
Objetivos	Garantir reservas bem-sucedidas, facilitar modificações e cancelamentos
Atividades	Selecionar datas, tipos de quarto, inserir informações pessoais, confirmar reserva
Questões	Disponibilidade de datas, políticas de cancelamento
Barreiras	Interface desatualizada, falta de informações claras
Funcionalidades	Interface moderna e intuitiva, atualização em tempo real de disponibilidade
Interação	Cliente interage com a interface do sistema
Mensagem	Facilidade, conveniência, transparência nas políticas
Onde Ocorre	Interface web/mobile do sistema de reservas
Tarefas Aparentes	Inserção de dados de reserva, seleção de datas e quartos
Tarefas Escondidas	Verificação de disponibilidade em tempo real, atualização de dados no banco de dados
Processos de Suporte	Suporte ao cliente, notificações de confirmação de reserva
Saída Desejável	Reserva concluída com sucesso, cliente informado e satisfeito

**Tabela 02: Notificações em Tempo Real**

Funcionalidade	Detalhes
Ações do Cliente	Receber notificações sobre reservas, atualizações de status de quartos
Objetivos	Manter o cliente informado
Atividades	Envio de notificações por email
Questões	Melhor forma de notificar o cliente, garantir leitura das notificações
Barreiras	Cliente não recebe ou ignora notificações, notificações mal formatadas
Funcionalidades	Sistema de notificações integrado com serviço de email
Interação	Cliente recebe notificações em seu dispositivo e pode interagir com elas
Mensagem	Atenção ao cliente, proatividade na comunicação
Onde Ocorre	Email do cliente
Tarefas Aparentes	Recebimento de notificações por email
Tarefas Escondidas	Geração automática de emails, monitoramento de entregas de notificações
Processos de Suporte	Notificações de confirmação, atualização de status
Saída Desejável	Cliente bem informado e satisfeito

**Tabela 03: Gestão de Quartos**

Funcionalidade	Detalhes
Ações do Cliente	Gerenciar disponibilidade e manutenção de quartos
Objetivos	Maximizar a ocupação e garantir a qualidade dos quartos
Atividades	Atualizar status de quartos, agendar manutenções
Questões	Monitoramento em tempo real, previsão de demandas
Barreiras	Sistemas legados, processos manuais
Funcionalidades	Painel de controle de disponibilidade, alertas de manutenção
Interação	Interface do gerente com o sistema
Mensagem	Eficiência, controle, proatividade
Onde Ocorre	Painel administrativo web
Tarefas Aparentes	Atualização de status, planejamento de manutenção
Tarefas Escondidas	Integração de dados
Processos de Suporte	Manutenção preventiva, gestão de equipe

Saída Desejável	Quartos sempre prontos e em boas condições, maximização da ocupação
-----------------	---------------------------------------------------------------------

**Proposta de Valor**

A Proposta de Valor do nosso projeto é uma arquitetura de referência para um sistema de reservas de hotel moderno e escalável, utilizando microserviços para melhorar eficiência operacional, experiência do usuário e resolver desafios de sistemas legados e comunicação."



### 1.1.4 Hipóteses

Com todo material coletado até o momento, foi possível estruturar as informações para gerar um grupo de insights que irão auxiliar no direcionamento do desenvolvimento do projeto. A tabela de observações de hipóteses é uma forma de sintetizar o material gerado para facilitar a priorização das atividades que precisam ser realizadas.

**Tabela de Observações de Hipóteses**

Observação	Hipótese
Muitos hotéis ainda utilizam sistemas legados que são difíceis de manter e integrar com novas tecnologias.	Adotar uma arquitetura de microserviços para facilitar a manutenção e integração com novas tecnologias.
A falta de integração entre diferentes serviços pode resultar em comunicação ineficaz com os clientes.	Implementar um microserviço de notificações para garantir uma comunicação em tempo real com os clientes.
Sistemas tradicionais podem enfrentar dificuldades para escalar com o aumento da demanda.	Utilizar uma arquitetura escalável que possa lidar com um aumento no volume de reservas sem comprometer o desempenho.
Interfaces desatualizadas podem resultar em uma experiência de usuário ruim.	Desenvolver uma interface moderna e intuitiva utilizando um SPA (Single Page Application) para melhorar a experiência do usuário.
Manutenção manual dos registros de disponibilidade de quartos pode levar a erros e duplicações.	Automatizar o gerenciamento de disponibilidade de quartos para reduzir erros e duplicações.
Processamento de reservas manualmente pode ser demorado e propenso a erros.	Implementar um sistema de processamento automático de reservas para aumentar a eficiência e precisão.
A experiência de check-in/check-out pode ser lenta e ineficiente.	Adotar um sistema de check-in/check-out digital para agilizar o processo e melhorar a satisfação dos clientes.
Falta de visibilidade sobre o desempenho do sistema e problemas técnicos.	Implementar ferramentas de monitoramento e logging para fornecer visibilidade em tempo real sobre o desempenho e problemas do sistema.

## Ideias

Nesta seção, apresentamos ideias fundamentais para a arquitetura de referência do sistema de reservas de hotel, focando em melhorias operacionais, experiência do usuário e escalabilidade. Cada ideia é avaliada com critérios específicos para facilitar a priorização e direcionamento do projeto.

### Balizadores para Notas da Matriz

Escala	B Benefícios	A Abrangência	S Satisfação	I Investimentos	C Cliente	O Operacionalidade
5	De vital importância	Total (de 70% a 100%)	Muito grande	Alto investimento	Impacto muito grande no cliente	Muito fácil
4	Significativo	Muito grande (de 40% a 70%)	Grande	Médio investimento	Grande impacto	Fácil
3	Razoável	Razoável (de 20% a 40%)	Média	Alguns investimentos	Média facilidade	Médio
2	Pequena	Pequenos	Pequena	Pouco investimento	Pequeno impacto	Difícil
1	Alguns benefícios	Quase não é notada	Pouquíssimo	Muito pouco investimento	Nenhum impacto	Muito difícil

### Tabela de Ideias com Pontuação Detalhada

Ideias	Critérios de comparação					Total	Priorização
	B	A	S	I	O		
Arquitetura de Microserviços para otimizar a manutenção e integração de sistemas	4	4	4	3	5	20	1
Implementação de Microserviço de Notificações em Tempo Real para comunicação ágil	3	4	4	3	4	18	2
Desenvolvimento de Interface Moderna em SPA (Single Page Application) para melhor experiência do usuário	3	3	4	3	4	17	3
Automatização do Gerenciamento de Disponibilidade para eficiência operacional	3	3	3	4	3	16	4
Implementação de Sistema de Processamento Automático de Reservas para agilidade	3	3	3	3	3	15	5
Desenvolvimento de Check-in/Check-out digital para facilidade de uso	3	3	3	2	3	14	6
Integração de Ferramentas de Monitoramento e Logging para controle de performance	2	2	3	2	3	12	7

## 1.2 Solução

### 1.2.1 Objetivo SMART

“Pretendemos criar uma arquitetura de referência **(S)** com a adoção de microserviços **(A)** no prazo de 2 meses **(T)**, reduzir em 40% o tempo gasto hoje **(M)** desde a concepção até a entrega em produção **(R)**, bem como reduzir em 70% os índices de qualidade, além de conquistar um aumento em 40% nos níveis de satisfação dos nossos clientes **(M)**.”

### 1.2.2 Premissas e Restrições

#### Premissas

- A escolha e adoção de tecnologias de microserviços (como Docker e Kubernetes) são viáveis e aceitas pela equipe técnica.
- A integração com sistemas legados será possível sem comprometer a estabilidade do sistema.
- A equipe de desenvolvimento terá acesso contínuo aos recursos necessários, como ambientes de desenvolvimento e ferramentas de monitoramento.
- As políticas de segurança e conformidade serão seguidas durante todo o ciclo de desenvolvimento e implantação.

#### Impactos das Premissas não Satisfeitas

- Se as tecnologias de microserviços não forem aceitas, a modularidade e escalabilidade planejadas podem não ser alcançadas, afetando diretamente a flexibilidade do sistema.
- Falhas na integração com sistemas legados podem resultar em interrupções no serviço e na experiência do usuário.
- A falta de recursos adequados pode atrasar o desenvolvimento e comprometer a qualidade do produto final.
- Violações de segurança podem expor o sistema a riscos significativos e penalidades regulatórias.



## Restrições

- Orçamento limitado para aquisição de novas tecnologias e recursos adicionais.
- Prazo de 2 meses para implementação da arquitetura de referência, com entrega incremental de funcionalidades.
- Conformidade com regulamentações de privacidade de dados e segurança cibernética.
- Disponibilidade limitada de equipe especializada em tecnologias específicas, como Docker e Kubernetes.

## Matriz de Risco

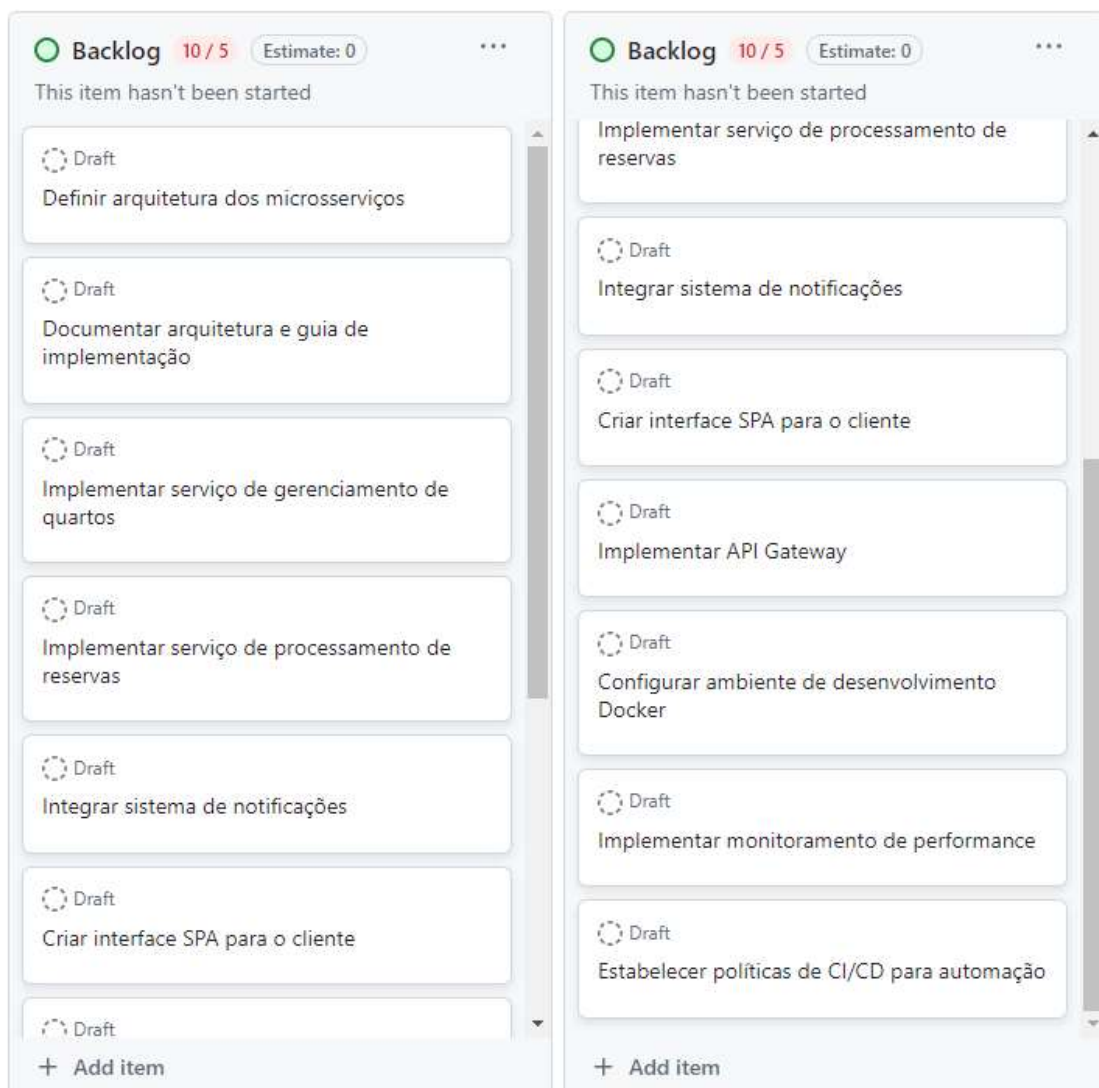
Riscos Potenciais	Probabilidade	Impacto	Deteção	Severidade	Ações de Mitigação
Aceitação limitada das tecnologias de microserviços	Média	Alto	Média	Alta	Realizar workshops e sessões de treinamento para familiarizar a equipe com as novas tecnologias.
Falhas na integração com sistemas legados	Alta	Alto	Alta	Muito alta	Realizar testes extensivos de integração e planos de contingência para fallback.
Recursos inadequados para desenvolvimento	Baixa	Médio	Alta	Média	Revisar e ajustar o planejamento de recursos com base na carga de trabalho e prioridades.
Violações de segurança e conformidade	Alta	Muito alto	Alta	Muito alta	Implementar políticas de segurança rigorosas e realizar auditorias regulares.
Orçamento insuficiente para tecnologias adicionais	Média	Médio	Média	Alta	Priorizar recursos essenciais e buscar alternativas de financiamento se necessário.

## Explicação dos Campos

- **Probabilidade:** Estima a chance de o risco ocorrer (Baixa, Média, Alta).
- **Impacto:** Avalia o impacto no projeto caso o risco se materialize (Baixo, Médio, Alto, Muito alto).
- **Detecção:** Nível de facilidade de detecção do risco antes que ele se torne um problema sério (Baixa, Média, Alta).
- **Severidade:** Combinação de Impacto e Probabilidade para priorizar riscos (Baixa, Média, Alta, Muito alta).
- **Ações de Mitigação:** Estratégias ou ações para reduzir a probabilidade ou impacto do risco.

### 1.2.3 Backlog de Produto

Nesta seção, apresentamos o backlog de produto detalhado para o projeto de arquitetura de referência do sistema de reservas de hotel. Utilizamos o [GitHub Boards](#) como plataforma central para gerenciar e organizar todas as atividades e requisitos essenciais para o desenvolvimento da solução. O GitHub Boards proporcionou uma abordagem colaborativa e transparente, permitindo uma gestão eficiente das tarefas ao longo das iterações do projeto.

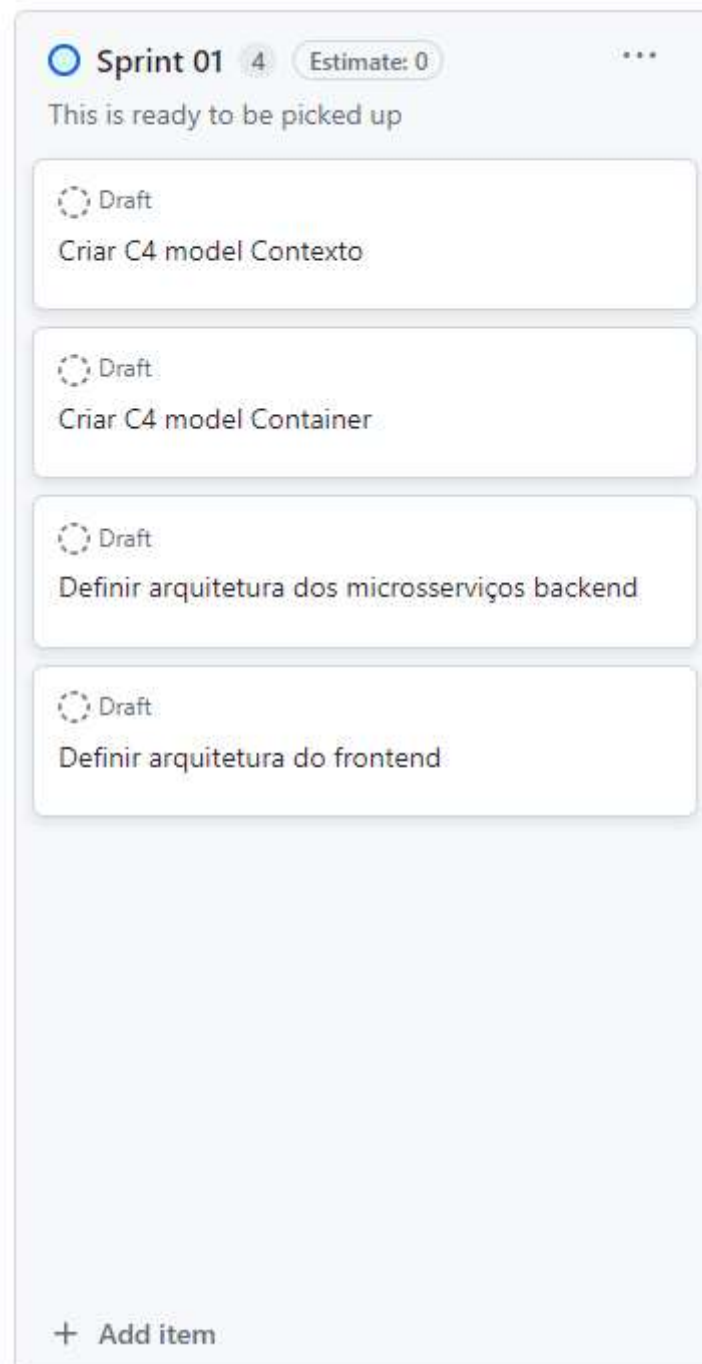


## 2. Área de Experimentação

### 2.1 Sprint 1

#### 2.1.1 Solução

- Evidência do planejamento:




- Evidência da execução de cada requisito:

## Criar C4 model Contexto

Veja abaixo a criação do C4 model nível contexto usando o [DSL](#) e [Structurizr](#).

Structurizr DSL language reference ▾

Upload </>  Render

```
1 workspace {
2
3   model {
4     viajante = person "Viajante Frequente" "Reserva quartos e gerencia estadias."
5     gerente = person "Gerente de Hotel" "Gerencia as operações do hotel."
6
7     sistema = softwareSystem "Projeto Aplicado" "Sistema de Reservas de Hotel" {
8       viajante -> this "Usa para reservar quartos e gerenciar estadias"
9       gerente -> this "Usa para gerenciar operações do hotel"
10    }
11  }
12
13  views {
14    systemContext "Projeto Aplicado" {
15      include *
16      autolayout lr
17
18      person "Viajante Frequente" {
19        description "Reserva quartos e gerencia estadias."
20      }
21
22      person "Gerente de Hotel" {
23        description "Gerencia as operações do hotel."
24      }
25
26      softwareSystem "Projeto Aplicado" {
27        description "Sistema de Reservas de Hotel"
28      }
29    }
30
31    theme default
32  }
33 }
34
```

## Criar C4 model Container

Veja abaixo a criação do C4 model nível container usando o [DSL](#) e [Structurizr](#).

Structurizr DSL language reference ▼

```

1 workspace {
2
3   model {
4     user = person "Viajante Frequente" {
5       description "Usuário do sistema de reservas de hotel."
6     }
7
8     softwareSystem "Projeto Aplicado" {
9       description "Sistema de reservas de hotel."
10
11     webApp = container "SPA Web Application" {
12       description "Aplicação Single Page que permite aos usuários interagir com o sistema."
13       technology "React"
14     }
15
16     apiGateway = container "API Gateway" {
17       description "Gerencia e roteia as requisições para os microserviços apropriados."
18       technology "ASP.NET Core"
19     }
20
21     roomsService = container "Rooms Service" {
22       description "Gerencia as informações dos quartos."
23       technology "ASP.NET Core"
24     }
25
26     reservationsService = container "Reservations Service" {
27       description "Gerencia as reservas de quartos."
28       technology "ASP.NET Core"
29     }
30
31     notificationService = container "Notification Service" {
32       description "Envia notificações e alertas aos usuários."
33       technology "ASP.NET Core Background Service"
34     }
35
36     roomsDatabase = container "Rooms Database" {
37       description "Armazena os dados dos quartos."
38       technology "SQL Server"
39     }
40
41     reservationsDatabase = container "Reservations Database" {
42       description "Armazena os dados das reservas."
43       technology "SQL Server"
44     }
45
46     rabbitMQ = container "RabbitMQ" {
47       description "Gerencia a mensageria entre os microserviços."
48       technology "RabbitMQ"
49     }
50
51     user -> webApp "Utiliza"
52     webApp -> apiGateway "Faz requisições"
53     apiGateway -> roomsService "Roteia para"
54     apiGateway -> reservationsService "Roteia para"
55     apiGateway -> notificationService "Roteia para"
56     roomsService -> roomsDatabase "Lê e escreve dados em"
57     reservationsService -> reservationsDatabase "Lê e escreve dados em"

```

Structurizr DSL v2.2.0 - sor  
[DSL language reference](#)

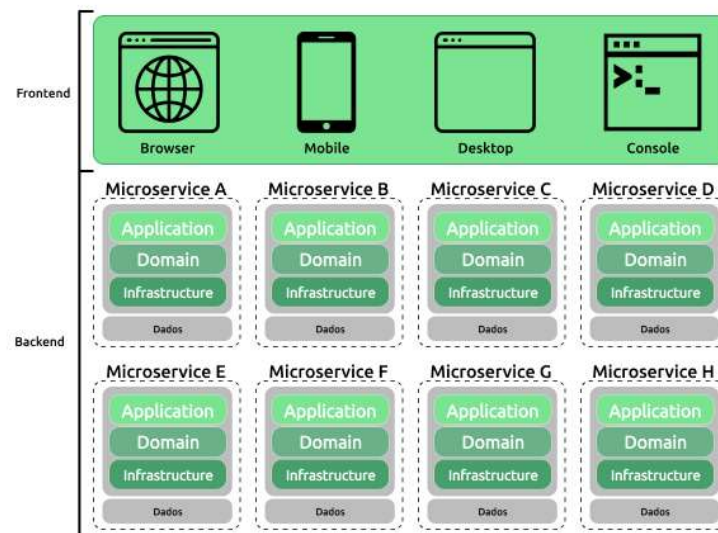
## Definir arquitetura dos microserviços backend

Para definir a arquitetura dos microserviços no backend, conduzi uma série de pesquisas.

Abaixo, apresento um [guia sobre microserviços](#) baseado nas informações do site [guia.dev](#).

## Microserviços

Arquitetura em microserviços, tema apresentado no guia, se tornou popular na última década e se consolidou como uma opção na construção de soluções. Na relação com o uso de camadas, os microserviços passam a ser uma divisão física orientada pelo domínio do negócio, e cada microserviço pode ainda aplicar o uso das camadas lógicas citadas no tema Camadas.



Link: <https://guia.dev/pt/pillars/software-architecture/layers-and-architecture-patterns.html#microservices>

## Definir arquitetura do frontend

O frontend será desenvolvido com Angular, seguindo o style guide da documentação oficial para garantir consistência e boas práticas.

Best Practices

### Angular coding style guide



Looking for an opinionated guide to Angular syntax, conventions, and application structure? Step right in. This style guide presents preferred conventions and, as importantly, explains why.

### Style vocabulary

Each guideline describes either a good or bad practice, and all have a consistent presentation.

The wording of each guideline indicates how strong the recommendation is.

**Do** is one that should always be followed. *Always* might be a bit too strong of a word. Guidelines that literally should always be followed are extremely rare. On the other hand, you need a really unusual case for breaking a *Do* guideline.

**Consider** guidelines should generally be followed. If you fully understand the meaning behind the guideline and have a good reason to deviate, then do so. Aim to be consistent.

**Avoid** indicates something you should almost never do. Code examples to *avoid* have an unmistakable red header.

#### Why?

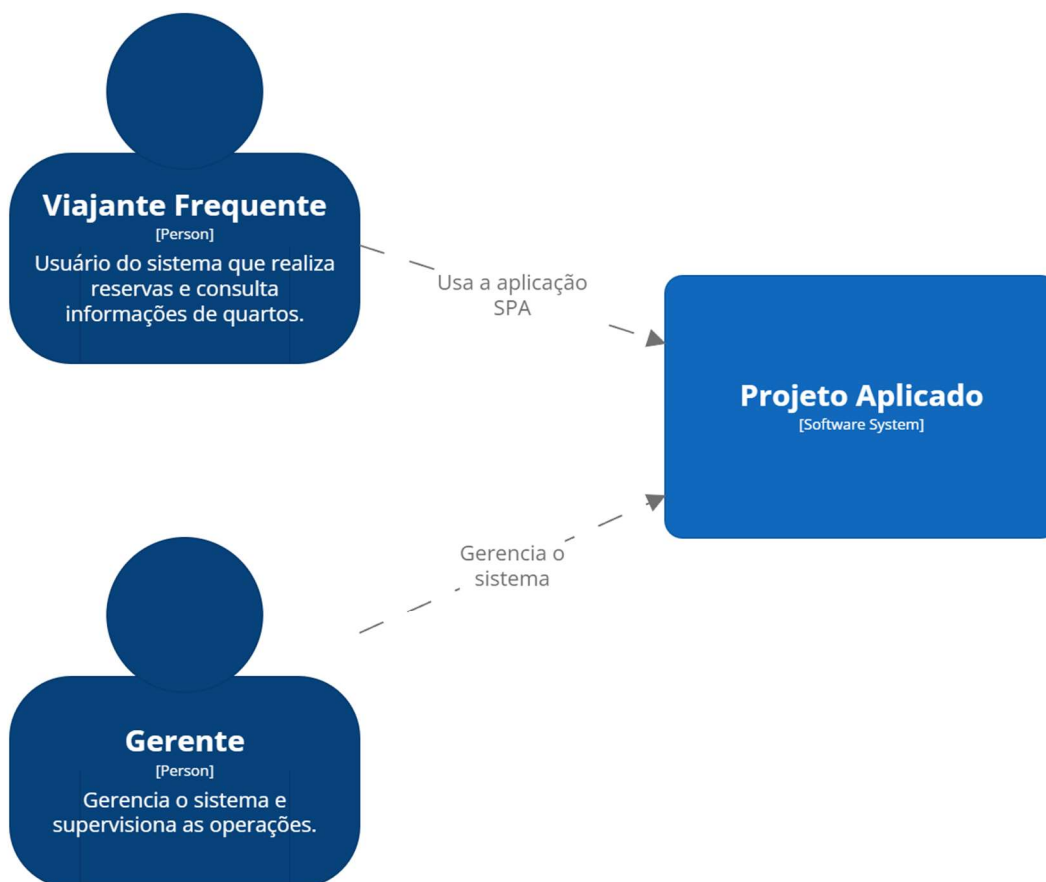
Gives reasons for following the previous recommendations.



- Evidência dos resultados:

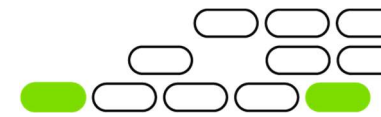
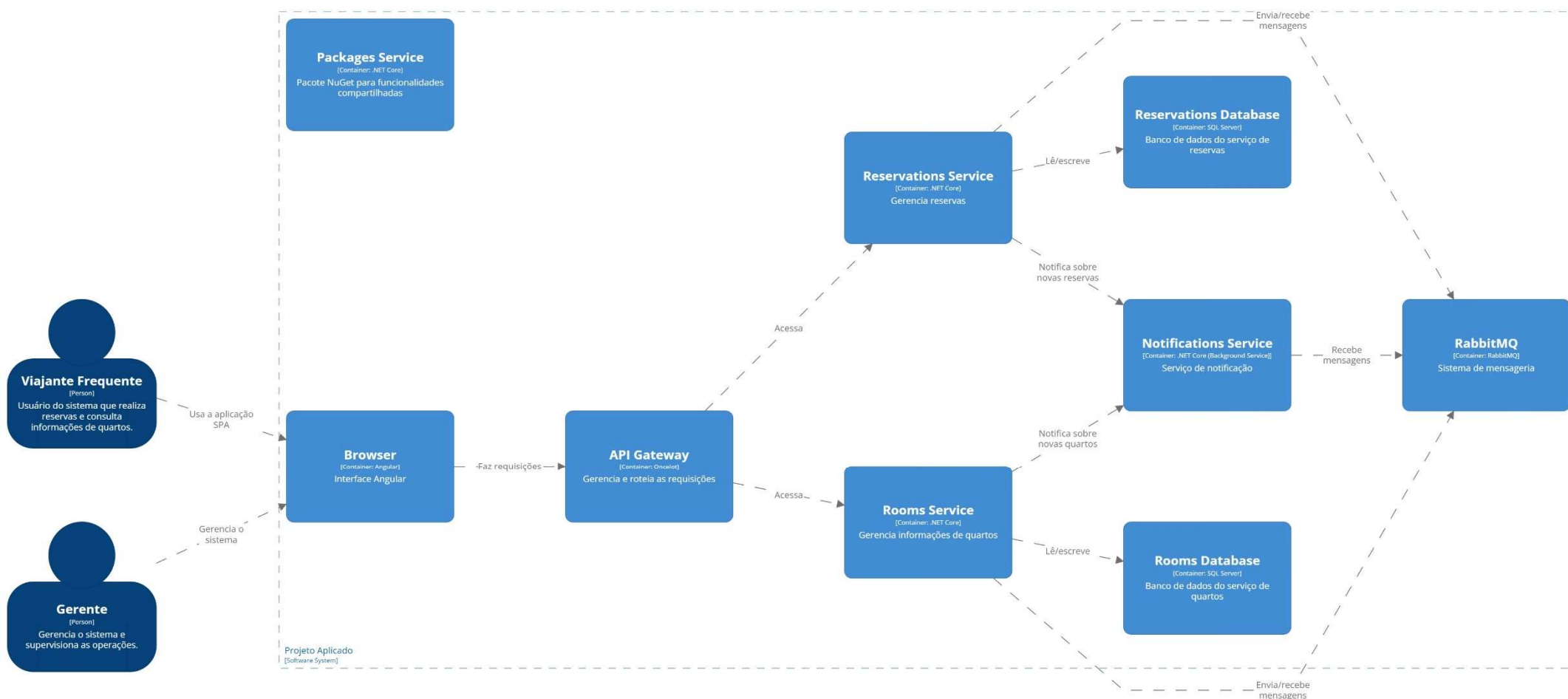
### Criar C4 model Contexto

A seguir, apresentamos o C4 Model no nível de contexto. Este diagrama fornece uma visão geral do sistema, destacando como ele interage com os usuários e outros sistemas externos. Ele ilustra o escopo do projeto e as principais entidades envolvidas, facilitando a compreensão do ambiente em que o sistema opera.



## Criar C4 model Container

A seguir, apresentamos o C4 Model no nível de container. Este diagrama detalha a estrutura interna do sistema, mostrando os principais containers e como eles se comunicam entre si e com os usuários.

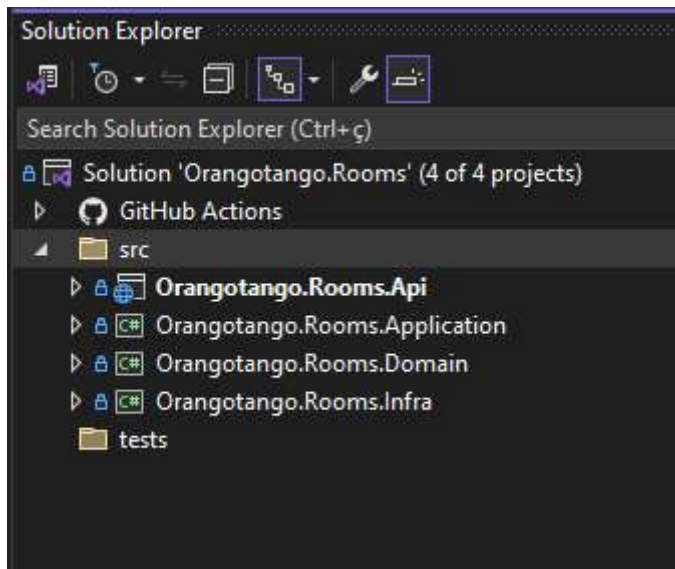


## Definir arquitetura dos microserviços backend

Os microserviços serão implementados em .NET 8 e seguirão uma Arquitetura Hexagonal / Orientada a Eventos (EDA). A arquitetura enfatiza a separação de responsabilidades, aderindo aos princípios SOLID, Clean Code e Domain-Driven Design (DDD). Os microserviços são organizados nas seguintes camadas:

- **Camada de Apresentação:** Contém APIs com controladores e modelos de entrada (DTOs) para definir contratos de endpoints.
- **Camada de Aplicação:** Inclui manipuladores de comandos, mapeadores, e serviços para processar a lógica de aplicação.
- **Camada de Domínio:** Abrange entidades, comandos, validações e abstrações de repositório que representam a lógica de negócios.
- **Camada de Infraestrutura:** Fornece implementações para repositórios e comunicação entre serviços, como barramentos de mensagens.

Veja a arquitetura em camadas do projeto de quartos.

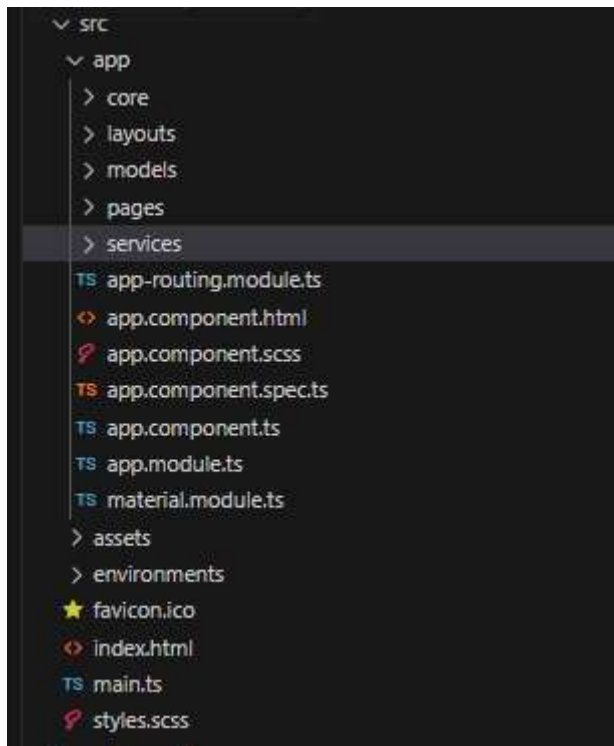


## Definir arquitetura do frontend

No front frontend seguiremos o style guide oficial do framework, que segue a estrutura abaixo de diretório.

- **/src/app:** Contém todos os arquivos principais do aplicativo.
- **/components:** Componentes reutilizáveis.
- **/services:** Serviços para lógica de negócios e comunicação com APIs.
- **/models:** Interfaces e modelos de dados.
- **/pages:** Componentes que representam páginas ou seções principais do aplicativo.
- **/src/assets:** Imagens, fontes e outros recursos estáticos.
- **/src/environments:** Arquivos de configuração para diferentes ambientes (desenvolvimento, produção).
- **/src/styles:** Arquivos de estilo global.

Veja abaixo a criação do projeto no Angular utilizando os diretórios acima.



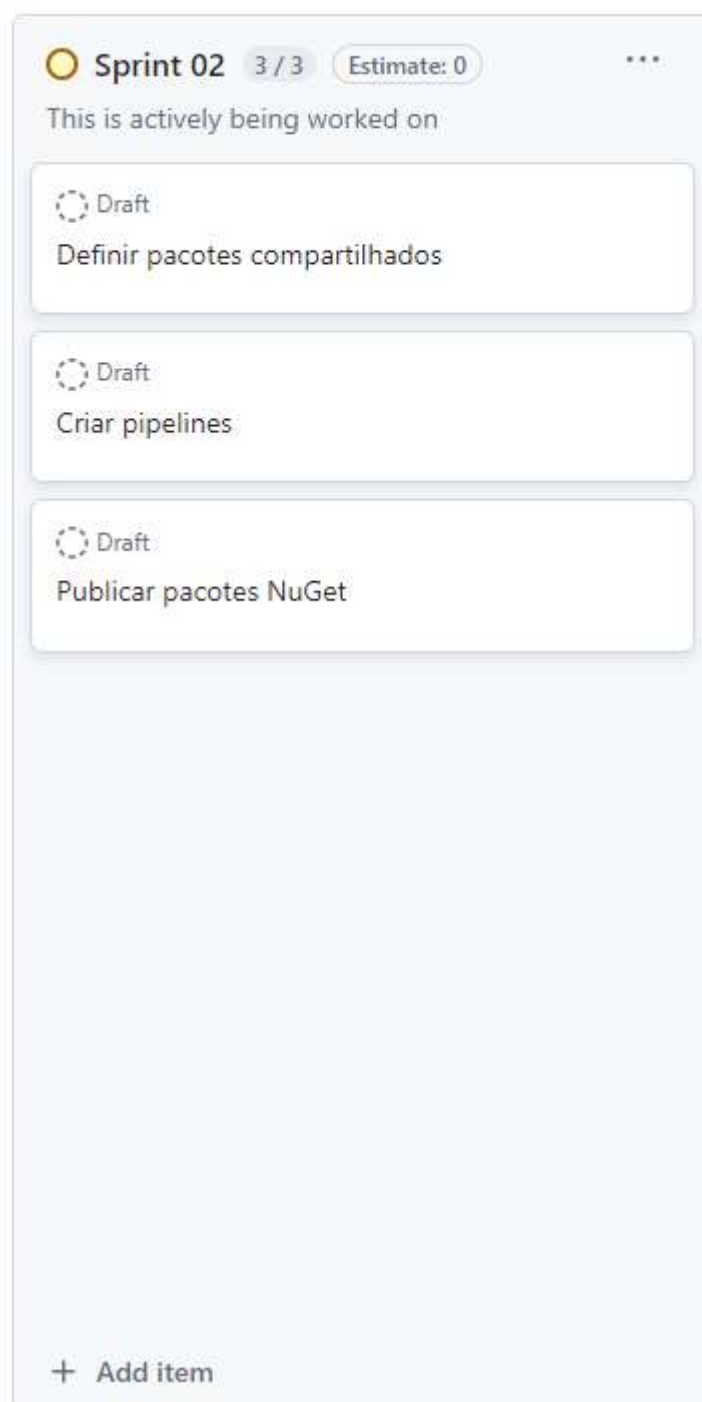
### 2.1.2 Lições Aprendidas

Durante a criação dos diagramas C4 utilizando o [Structurizr](#), notamos que a necessidade de escrever o código para gerar os diagramas tornou o processo mais desafiador e demorado. Apesar da flexibilidade que a codificação oferece, uma ferramenta que permita a criação dos diagramas diretamente pela interface gráfica seria mais eficiente e amigável. Isso facilitaria a visualização e a modificação dos diagramas, tornando o processo mais intuitivo e ágil. Avaliar ferramentas alternativas que oferecem uma interface gráfica para criação de diagramas C4 pode ser uma boa estratégia para futuras iterações.

## 2.2 Sprint 2

### 2.2.1 Solução

- Evidência do planejamento:



- Evidência da execução de cada requisito:

### Definir pacotes compartilhados

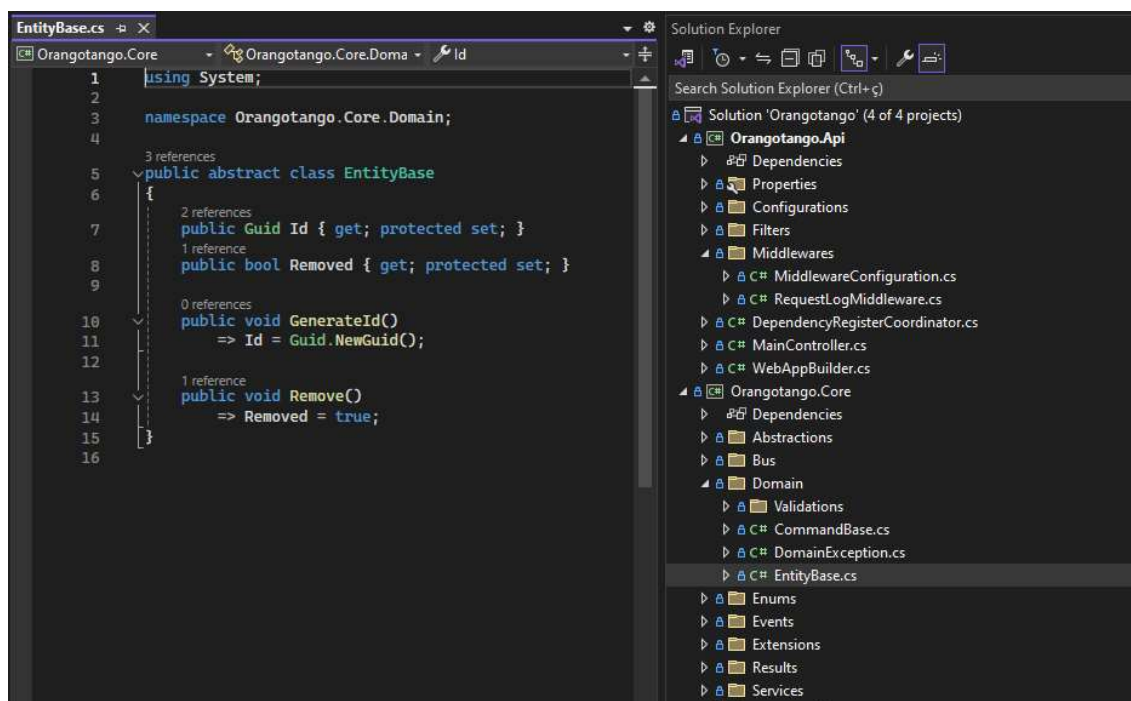
No contexto de projetos complexos, a criação de pacotes compartilhados é uma prática recomendada para garantir a reutilização de código e a padronização de funcionalidades comuns entre diferentes microsserviços, garantindo assim consistência e evitando a duplicação de código.

O principal objetivo ao criar pacotes compartilhados é encapsular funcionalidades que são comuns a vários microsserviços, como validações, manipulação de dados, integração com APIs externas, e até mesmo regras de negócio. Isso facilita a manutenção e a evolução do projeto, pois qualquer mudança necessária pode ser realizada em um único ponto, propagando-se automaticamente para todos os serviços que utilizam o pacote.

Segue a descrição dos pacotes compartilhados criados para o projeto:

- **Core:** O pacote central para microsserviços, abrangendo abstrações para mensagens, eventos, repositórios, serviços, agregações e mais.
- **Api:** Este pacote fornece APIs centralizando a configuração do Swagger, padrões de resposta padronizados, middleware para registro de logs de requisições recebidas e respostas enviadas.
- **Events:** Contém todos os eventos da solução, estabelecendo contratos para fácil integração via topologia de mensagens, facilitando a replicação de dados entre microsserviços.
- **Infra:** Inclui configurações relacionadas à infraestrutura, como contextos do Entity Framework, configuração de logs baseados em ELK, mensageria utilizando Mass Transit e RabbitMQ, entre outros.

Veja abaixo a criação da entidade base no pacote Core.





## Criar pipeline

Para a criação da pipeline, foi utilizado o GitHub Actions. A pipeline configurada é dividida em duas etapas:

1. **Build and Tests:** Esta etapa realiza o checkout do código, configura o ambiente .NET, restaura as dependências, compila o projeto e executa os testes unitários. Ela é executada automaticamente a cada push na branch "main".
2. **Publish Docker Hub:** Após a conclusão bem-sucedida da etapa de build e testes, essa etapa faz login no Docker Hub, constrói a imagem Docker da aplicação, e a publica no Docker Hub.

```

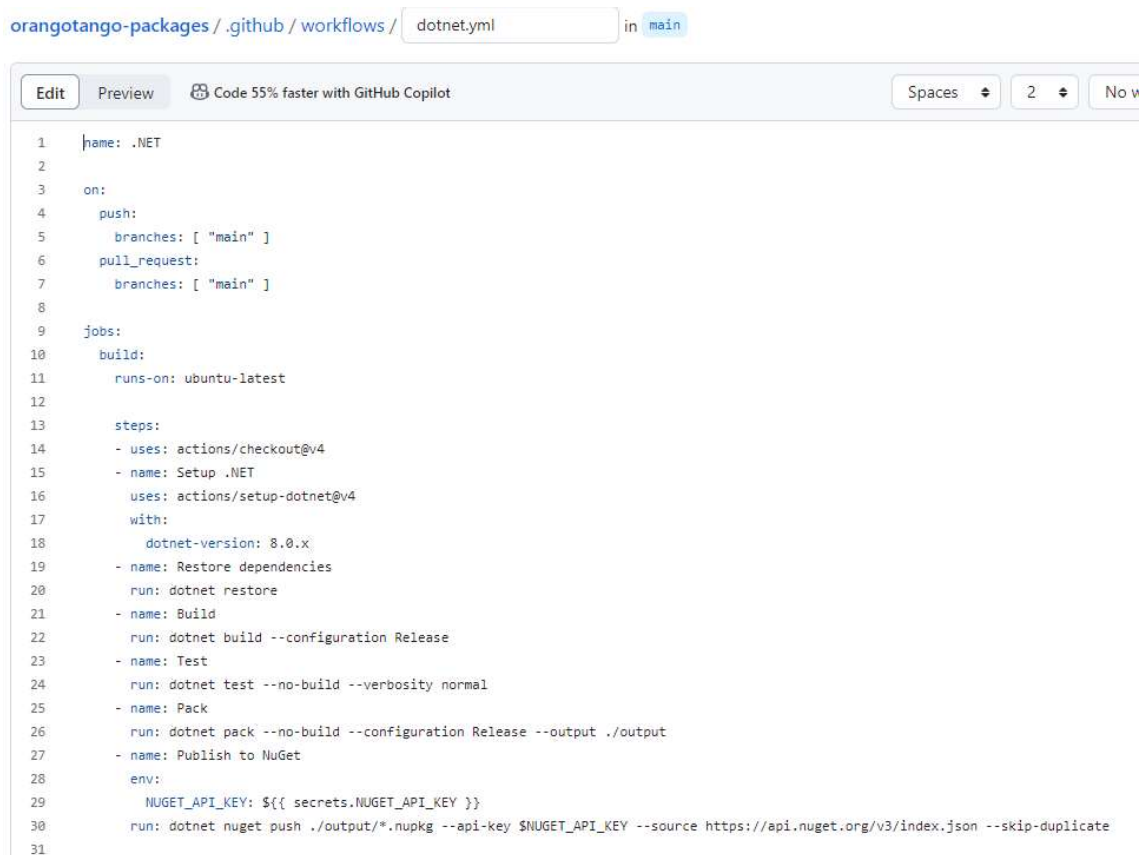
Code Blame 58 lines (52 loc) · 1.56 KB Code 55% faster with GitHub Copilot
6
7 jobs:
8   build-and-tests:
9     name: Build and Tests
10    runs-on: ubuntu-latest
11
12    steps:
13      - uses: actions/checkout@v4
14      - name: Setup .NET
15        uses: actions/setup-dotnet@v4
16        with:
17          dotnet-version: 8.0.x
18      - name: Restore dependencies
19        run: dotnet restore
20      - name: Build
21        run: dotnet build --no-restore
22      - name: Test
23        run: dotnet test --no-build --verbosity normal
24
25   publish-image:
26     runs-on: ubuntu-latest
27     needs: build-and-tests
28     name: Publish Docker Hub
29     steps:
30       - uses: actions/checkout@v3
31       - name: Login to Docker Hub
32         run: |
33           echo "${DOCKER_PASSWORD}" | docker login --username $DOCKER_USER --password-stdin
34           mkdir -p ~/.docker
35           cat <<EOF > ~/.docker/config.json
36           {
37             "auths": {
38               "https://index.docker.io/v1/": {
39                 "auth": "$(echo -n $DOCKER_USER:$DOCKER_PASSWORD | base64)"
40               }
41             },
42             "credsStore": "secretservice"
43           }
44           EOF
45     env:
46       DOCKER_USER: ${ secrets.DOCKER_USER }

```

## Publicar pacotes no NuGet

Para a criação das pipelines, foi utilizado o GitHub Actions, uma ferramenta de automação que permite configurar, testar e implantar projetos diretamente a partir do repositório GitHub. O pipeline configurado realiza o checkout do código, define o ambiente .NET, restaura as dependências, compila o projeto, executa os testes, empacota a aplicação e, finalmente, publica o pacote no NuGet.

Veja YML abaixo:



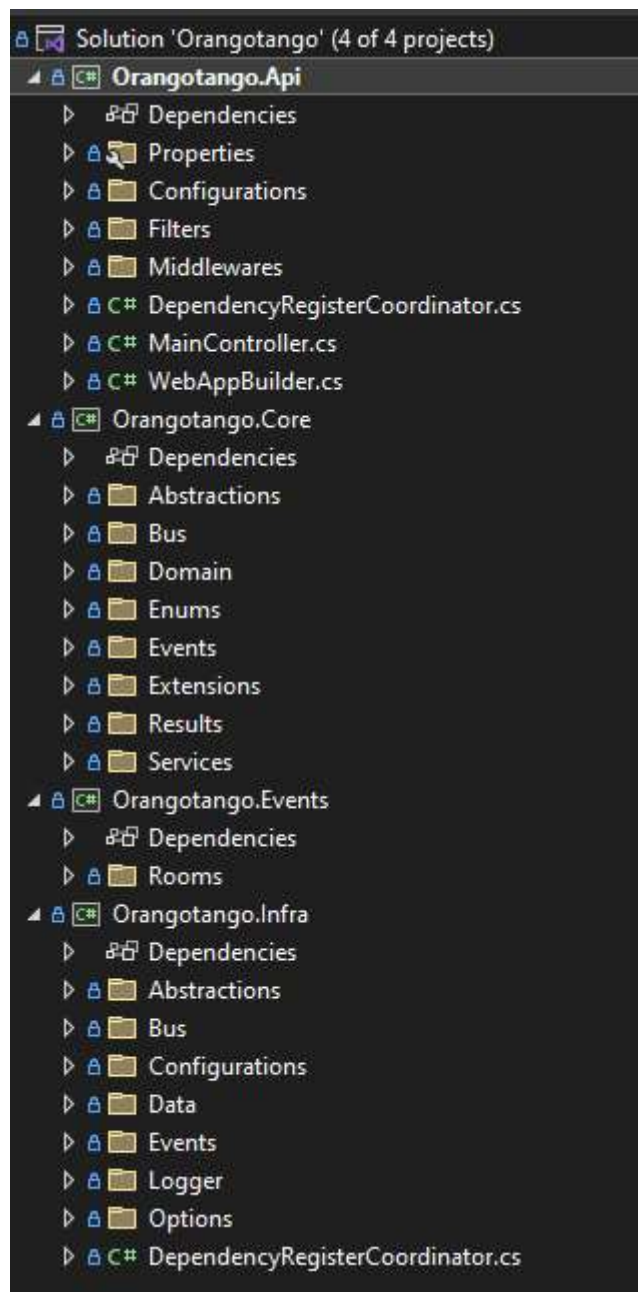
The screenshot shows a GitHub Actions workflow file named `dotnet.yml` in the `orangotango-packages` repository. The workflow is configured to run on the `main` branch. It includes a `build` job that runs on `ubuntu-latest`. The job has several steps: `checkout` (actions/checkout@v4), `Setup .NET` (actions/setup-dotnet@v4), `Restore dependencies` (dotnet restore), `Build` (dotnet build --configuration Release), `Test` (dotnet test --no-build --verbosity normal), `Pack` (dotnet pack --no-build --configuration Release --output ./output), and `Publish to NuGet` (dotnet nuget push ./output/\*.nupkg --api-key \$NUGET\_API\_KEY --source https://api.nuget.org/v3/index.json --skip-duplicate). The workflow also includes an `env` block with `NUGET_API_KEY` set to `${{ secrets.NUGET_API_KEY }}`.

```
1 name: .NET
2
3 on:
4   push:
5     branches: [ "main" ]
6   pull_request:
7     branches: [ "main" ]
8
9 jobs:
10  build:
11    runs-on: ubuntu-latest
12
13    steps:
14      - uses: actions/checkout@v4
15      - name: Setup .NET
16        uses: actions/setup-dotnet@v4
17        with:
18          dotnet-version: 8.0.x
19      - name: Restore dependencies
20        run: dotnet restore
21      - name: Build
22        run: dotnet build --configuration Release
23      - name: Test
24        run: dotnet test --no-build --verbosity normal
25      - name: Pack
26        run: dotnet pack --no-build --configuration Release --output ./output
27      - name: Publish to NuGet
28        env:
29          NUGET_API_KEY: ${ secrets.NUGET_API_KEY }
30        run: dotnet nuget push ./output/*.nupkg --api-key $NUGET_API_KEY --source https://api.nuget.org/v3/index.json --skip-duplicate
31
```

- Evidência dos resultados:

### Definir pacotes compartilhados

Veja abaixo a criação dos pacotes utilizando .NET 8

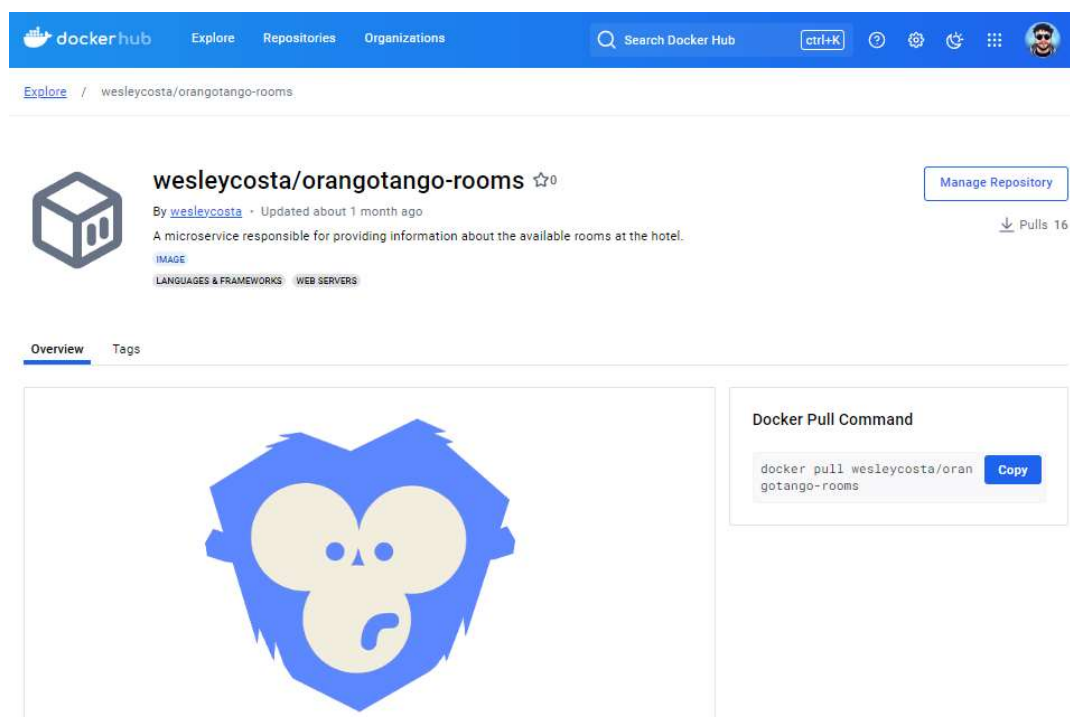


## Criar pipeline

Veja abaixo a execução com sucesso da pipeline.



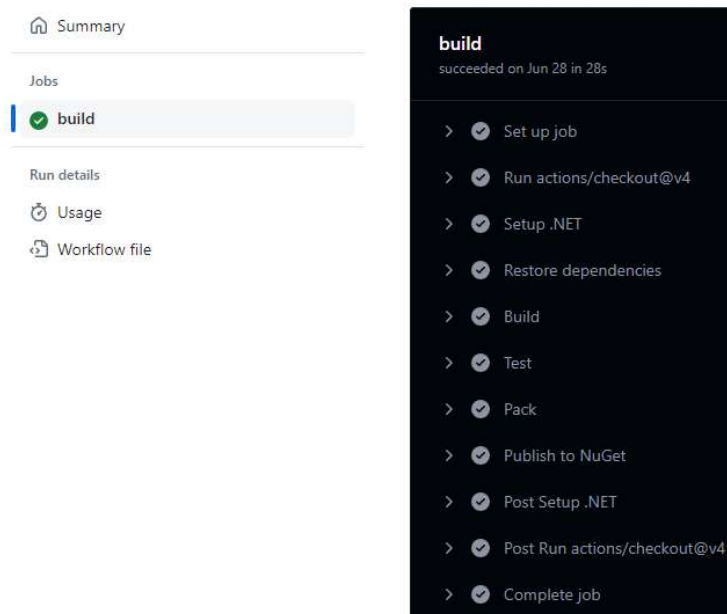
Veja abaixo a publicação da imagem no Docker Hub.



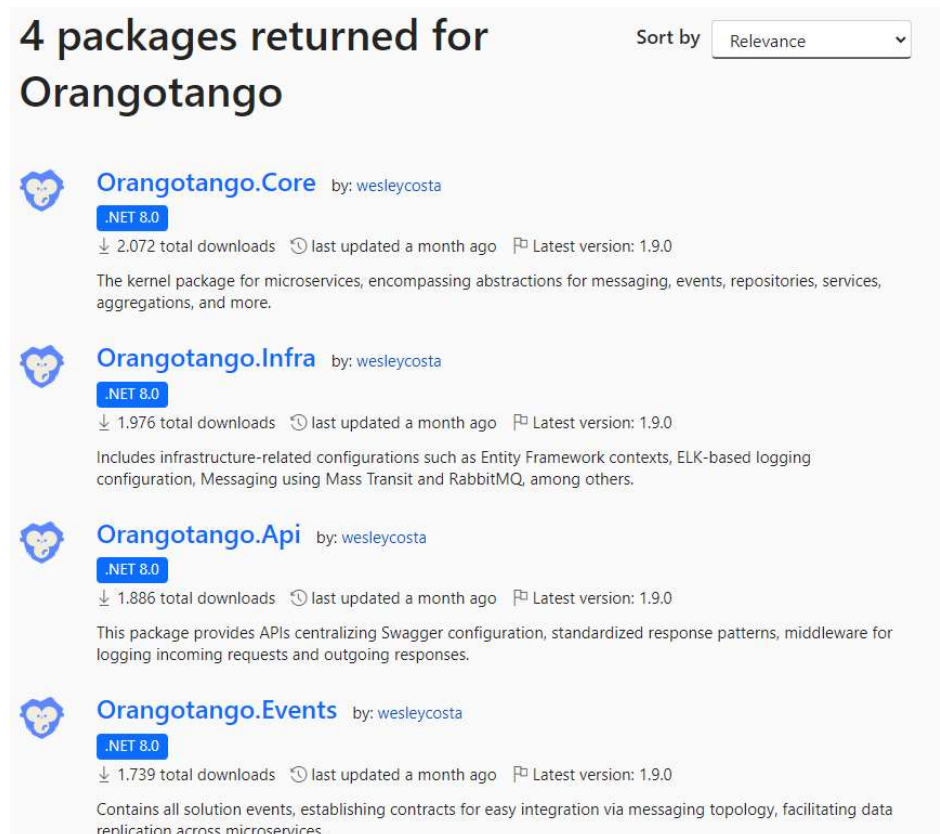
Link da imagem: <https://hub.docker.com/r/wesleycosta/orangotango-rooms>

## Publicar pacotes no Nuget

Veja abaixo a execução com sucesso da pipeline.



Veja abaixo a publicação dos pacotes no Nuget.



Link da imagem: <https://www.nuget.org/packages?q=Orangotango>

### 2.2.2 Lições Aprendidas

Durante o processo de publicação de imagens de cada microserviço no Docker Hub, foi necessário criar secrets individuais (DOCKER\_USER e DOCKER\_PASSWORD) em cada repositório. No entanto, uma possível melhoria seria investigar formas de centralizar essas informações em um único lugar, permitindo sua reutilização em outros repositórios. Isso pode simplificar a gestão de credenciais e reduzir a repetição de configuração.

## 2.3 Sprint 3

### 2.3.1 Solução

- Evidência do planejamento:

**Sprint 03**

2 / 5

Estimate: 0

...

This item is in review

Draft

Definir monitoramento

Draft

Definir observabilidade

+ Add item

- Evidência da execução de cada requisito:

### Definir monitoramento

Para microserviços, é extremamente importante ter logs e rastreamento entre os diferentes serviços. Uma estratégia eficaz é utilizar a pilha ELK, com o Kibana para exibir e visualizar os logs de maneira centralizada.

Veja abaixo a configuração da aplicação usando o Serilog para salvar os logs no Elasticsearch.

```

9
10 0 references
11  internal static class SerilogConfiguration
12  {
13      1 reference
14      public static IServiceCollection AddSerilog(this IServiceCollection services,
15      IConfiguration configuration)
16      {
17          var options = configuration.GetElasticsearchOptions();
18          var elasticsearchUri = new Uri(options.Uri);
19
20          var elasticsearchSinkOptions = new ElasticsearchSinkOptions(elasticsearchUri)
21          {
22              AutoRegisterTemplate = true,
23              AutoRegisterTemplateVersion = AutoRegisterTemplateVersion.ESv7,
24              IndexFormat = options.IndexFormat
25          };
26
27          Log.Logger = new LoggerConfiguration()
28              .Enrich
29              .FromLogContext()
30              .WriteTo
31              .Elasticsearch(elasticsearchSinkOptions)
32              .WriteTo
33              .Console()
34              .CreateLogger();
35
36          return services.AddSingleton(Log.Logger);
37      }
38
39      1 reference
40      private static ElasticsearchOptions GetElasticsearchOptions(this IConfiguration configuration)
41      {
42          var options = new ElasticsearchOptions();
43          configuration.Bind(ElasticsearchOptions.Elasticsearch, options);
44
45          return options;
46      }
47  }

```



## Definir observabilidade

Além dos logs, é crucial estabelecer uma estratégia de observabilidade para os microserviços. Isso facilita a identificação de gargalos operacionais e permite monitorar a saúde geral da aplicação, promovendo maior eficiência e proatividade na resolução de problemas. Para isso, utilizaremos o New Relic como APM (Application Performance Monitoring).

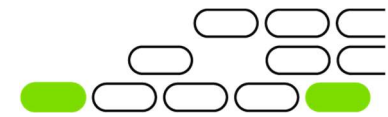
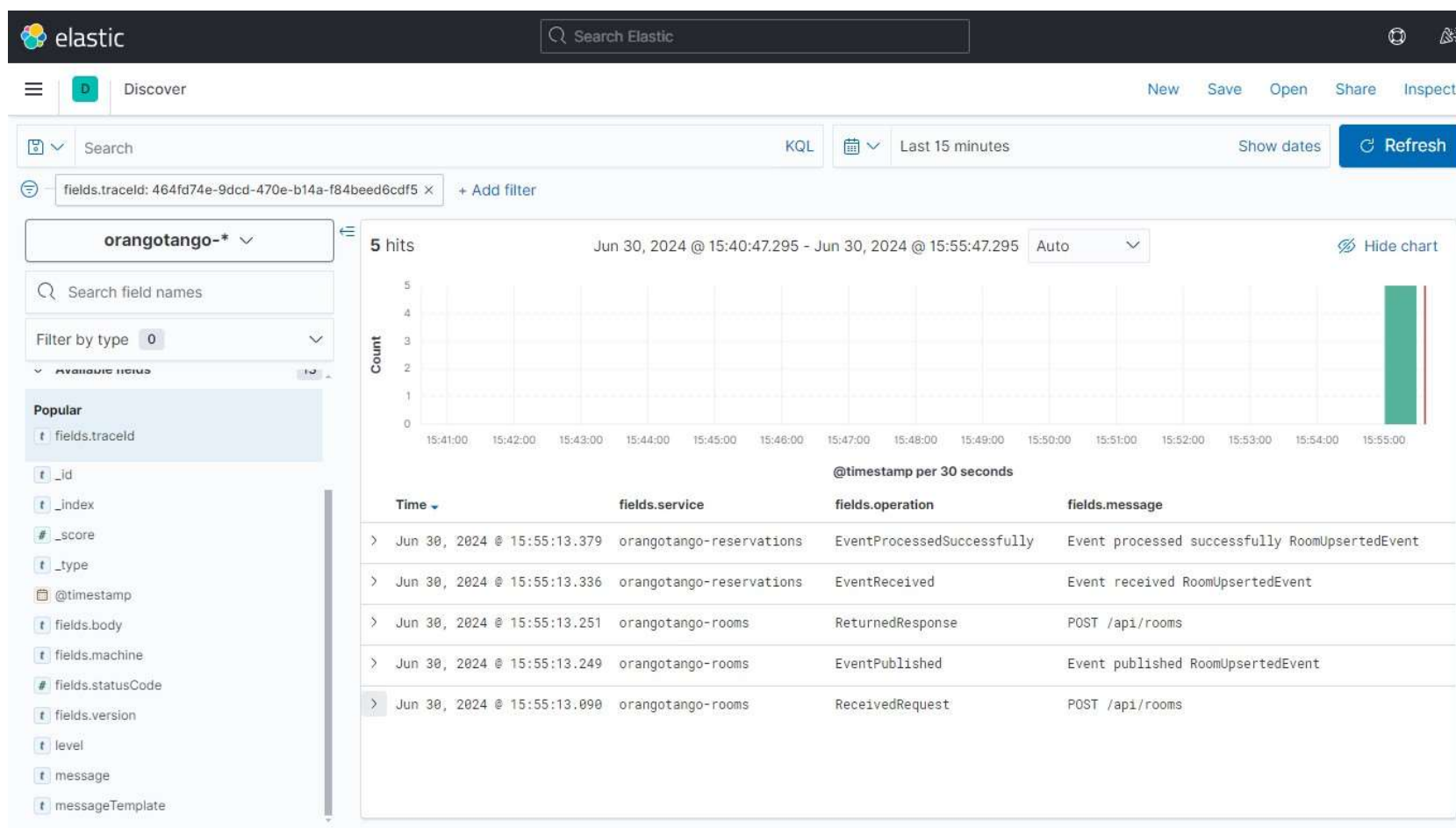
Veja abaixo a configuração do agente New Relic para .NET no Dockerfile.

```
C: > Projetos > orangotango-rooms > Dockerfile > ...
1 # Stage 1: Build the application
2 FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
3 WORKDIR /app
4
5 # Copy and restore dependencies
6 COPY . ./
7 COPY src/Orangotango.Rooms.Api/Orangotango.Rooms.Api.csproj ./src/
8 RUN dotnet restore
9
10 # Publish the application
11 RUN dotnet publish -c Release -o out
12
13 # Stage 2: Build runtime image
14 FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS runtime
15 WORKDIR /app
16
17 # Copy built application from previous stage
18 COPY --from=build /app/out .
19
20 # Install New Relic agent
21 RUN apt-get update && apt-get install -y wget ca-certificates gnupg \
22     && echo 'deb http://apt.newrelic.com/debian/ newrelic non-free' | tee /etc/apt/sources.list.d/newrelic.list \
23     && wget https://download.newrelic.com/548C16BF.gpg \
24     && apt-key add 548C16BF.gpg \
25     && apt-get update \
26     && apt-get install -y newrelic-dotnet-agent \
27     && rm -rf /var/lib/apt/lists/*
28
29 # Set environment variables for New Relic agent
30 ENV CORECLR_ENABLE_PROFILING=1 \
31     CORECLR_PROFILER={36032161-FFC0-4B61-B559-F6C5D41BAE5A} \
32     CORECLR_NEWRELIC_HOME=/usr/local/newrelic-dotnet-agent \
33     CORECLR_PROFILER_PATH=/usr/local/newrelic-dotnet-agent/libNewRelicProfiler.so \
34     NEW_RELIC_APP_NAME="orangotango-rooms"
35
36 # Set the entry point for the application
37 ENTRYPOINT ["dotnet", "Orangotango.Rooms.Api.dll"]
```

- Evidência dos resultados:

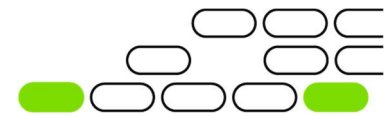
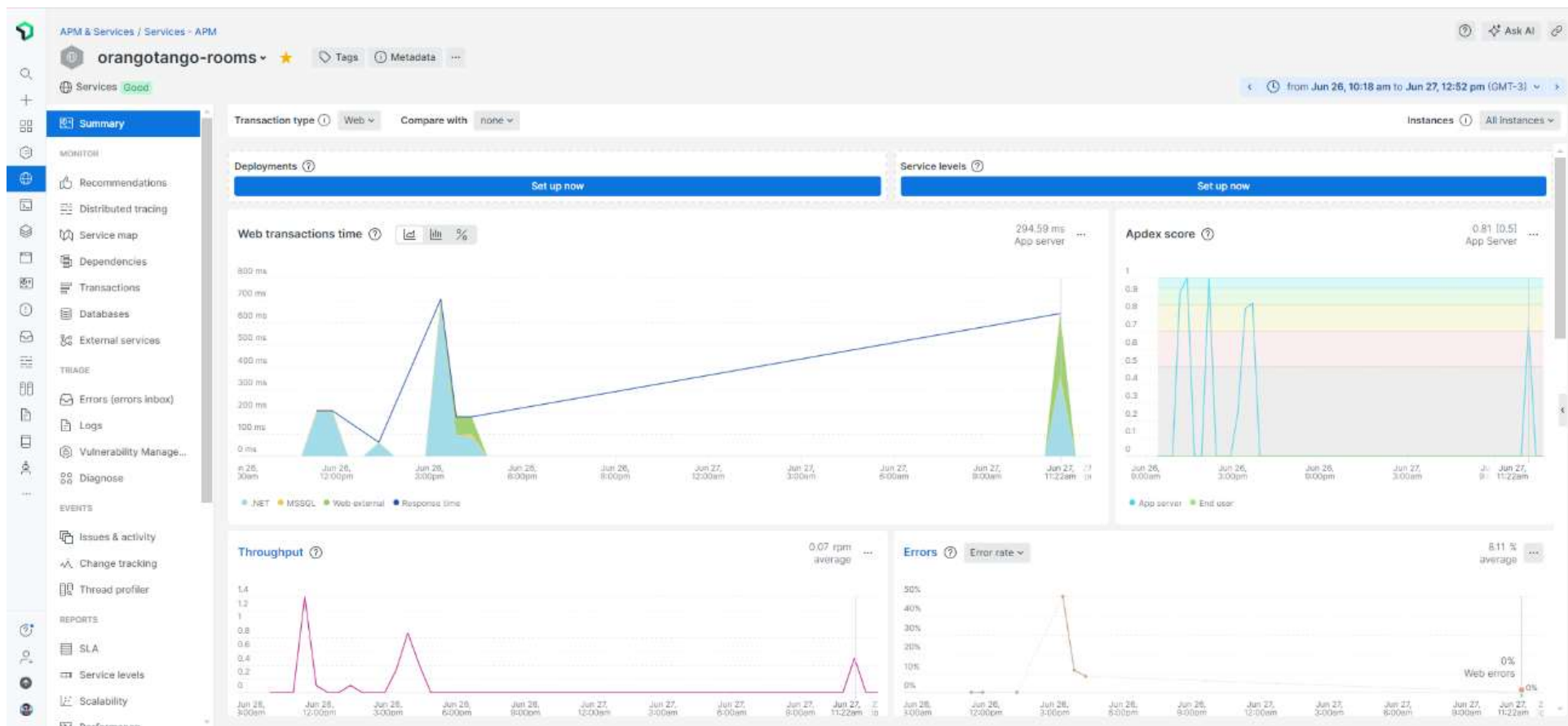
## Definir monitoramento

Abaixo estão os logs e o rastreamento entre os microsserviços durante o processo de criação de um quarto.



## Definir observabilidade

Confira abaixo o dashboard do New Relic com as transações, throughput, Apdex e erros.



### 2.3.2 Lições Aprendidas

A configuração do New Relic exige a instalação do agente .NET no Dockerfile. Essa instalação foi realizada em cada Dockerfile de cada microserviço individualmente.

No entanto, como uma melhoria futura, seria interessante criar uma imagem base com o SDK do .NET e a instalação do agente, centralizando o ponto de modificação e facilitando atualizações futuras.

## 3. Considerações Finais

### 3.1 Resultados

O Projeto Aplicado conseguiu alcançar resultados significativos no desenvolvimento e implementação de uma arquitetura de referência para um sistema de reservas de hotel moderno e escalável.

Entre os principais resultados obtidos, destacam-se:

- **Eficiência Operacional:** A adoção da arquitetura de microserviços permitiu uma manutenção mais ágil e uma integração eficiente entre diferentes serviços, reduzindo o tempo e os erros operacionais. A automatização de processos como o gerenciamento de disponibilidade de quartos e o processamento de reservas contribuiu para uma operação mais fluida e confiável.
- **Melhoria na Experiência do Usuário:** A implementação de uma interface moderna desenvolvida em Angular proporcionou uma experiência de usuário superior, com maior intuitividade e responsividade.
- **Escalabilidade:** A arquitetura escalável desenvolvida permitiu que o sistema atendesse a um aumento significativo no volume de reservas sem comprometer o desempenho. Essa capacidade de escalabilidade garantiu a continuidade do serviço durante picos de demanda, atendendo às necessidades crescentes do mercado.

## 3.2 Contribuições

O Projeto Aplicado trouxe diversas contribuições relevantes para a solução do desafio proposto na indústria hoteleira:

- **Inovação Tecnológica:** A adoção de uma arquitetura de microserviços representou uma inovação significativa, permitindo maior modularidade, flexibilidade e escalabilidade em comparação com sistemas monolíticos tradicionais. Essa abordagem facilitou a manutenção e a integração com novas tecnologias, atendendo às necessidades crescentes do mercado.
- **Escalabilidade e Desempenho:** A arquitetura escalável desenvolvida permitiu que o sistema de reservas acompanhasse o crescimento da demanda sem perda de desempenho, garantindo a continuidade do serviço durante picos de uso. Essa característica é uma vantagem competitiva importante para hotéis que desejam expandir seus serviços.
- **Redução de Custos Operacionais:** A automação de processos e a integração eficiente dos serviços reduziram os custos operacionais e de manutenção, proporcionando uma solução econômica e sustentável para os hotéis.
- **Contribuições Acadêmicas e Práticas:** O projeto agrega valor ao campo da arquitetura de software e sistemas de reservas, fornecendo um estudo de caso detalhado sobre a implementação de microserviços em um contexto real. Serve como referência para futuras pesquisas e desenvolvimentos na área, contribuindo para o avanço do conhecimento e das melhores práticas no setor.

### 3.3 Próximos passos

Para continuar o desenvolvimento do projeto Orangotango, foram identificados os seguintes próximos passos:

- **Publicação no Kubernetes:** Configurar e implementar a publicação dos microsserviços no Kubernetes, garantindo alta disponibilidade, escalabilidade, e facilidade de gerenciamento em ambiente de produção.
- **Melhoria na Configuração do New Relic:** Como melhoria futura, planeja-se criar uma imagem base que contenha o SDK .NET e o agente do New Relic, facilitando assim a atualização e a manutenção dos serviços.
- **Melhoria no CI/CD com GitHub Actions:** Expandir as pipelines de CI/CD utilizando o GitHub Actions para englobar processos adicionais de deploy, além de integrar o monitoramento de qualidade de código.
- **Adição de Resiliência à Aplicação:** Implementar padrões de resiliência como Circuit Breaker e Retry Pattern para melhorar a robustez e a capacidade da aplicação de lidar com falhas temporárias, evitando a propagação de erros e melhorando a experiência do usuário.