PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

PUC Minas Virtual

Pós-graduação *Lato Sensu* em Arquitetura de Soluções

Projeto Integrado

Relatório Técnico

Appointfy: Plataforma de oferta de serviços, agendamentos e engajamento com clientes

Rodrigo Garcia Topan Moreira

São Paulo
Janeiro 2023

Sumário

1. Introdução: Contexto da Solução

Atualmente, muitos pequenos comércios, como salões de beleza e barbeiros, possuem aplicativos ou websites para agendamentos de serviços, facilitando a vida dos clientes. Esses aplicativos geralmente são simples e possuem telas para cadastro de clientes e gestão administrativa. Eles também oferecem promoções para fidelização, como serviços gratuitos após uma certa quantidade de agendamentos. No entanto, esses aplicativos podem ter custos elevados, como a contratação de desenvolvedores e manutenção de sistemas, impedindo que pequenos comércios e empreendedores individuais os utilizem. Além disso, esses aplicativos geralmente não atraem novos clientes e se restringem a fidelizar a base de clientes existente.

Pensando nisso, propomos o desenvolvimento de uma plataforma web, similar aos grandes aplicativos de hotelaria como Airbnb e Booking, para democratizar o acesso a esses importantes aplicativos de agendamento de serviços. A plataforma permitiria o cadastro de diversos estabelecimentos e prestadores de serviços, fidelizando a base de clientes já existente e conectando esses estabelecimentos a novos clientes. Ela funcionaria como um marketplace de divulgação e agendamento de serviços.

Os clientes teriam a facilidade de encontrar estabelecimentos bem avaliados, comparar preços e usufruir de programas de fidelidade e descontos. Já os estabelecimentos poderiam contar com a divulgação e visibilidade da plataforma para alavancar suas vendas, teriam maior facilidade na gestão de suas atividades e pagariam apenas por agendamentos realizados.

Com isso, criamos um relacionamento ganha-ganha, onde quanto mais a plataforma alavanca as vendas dos nossos clientes, mais rentável e sustentável ela se torna.

1.1 Objetivos do Projeto

O objetivo deste projeto é projetar e implementar uma arquitetura ponta a ponta de uma plataforma de compartilhamento de serviços e agendamentos, que seja escalável, de custo relativamente baixo e com código de fácil manutenção.

Os objetivos específicos propostos são:

- Realizar um estudo sobre as regras de negócio envolvendo aplicativos de agendamento;
- Descrever os requisitos funcionais e não funcionais da aplicação de forma resumida, clara e objetiva;
- Definir a arquitetura utilizada e seus componentes de forma clara, utilizando dos diagramas arquiteturais mais populares (como diagrama de contexto, diagrama de containers e de componentes)
- Desenvolver um projeto de software utilizando as melhores práticas de design de sistemas (system design), como Clean Architecture, Arquitetura Hexagonal ou Domain Driven Design.
- Definir uma estratégia de implementação da arquitetura em nuvem, pensando em escalabilidade horizontal e alta disponibilidade
- Avaliar criticamente os resultados da arquitetura proposta

1.2 Canvas Modelo de Negócios

A seguir especifico através do Canvas, o modelo de negócio a ser seguido, identificando a proposta de valor, atividades chave, parceiros chave, os recursos chave, estrutura de custos, canais de relacionamento, formas de venda, segmento de clientes e fontes de receita. Identificar esses elementos é fundamental para a sustentabilidade de novos negócios.

1

[1] ***Figura - Modelo Canvas. Disponível em: https://github.com/RodrigoTopan/appointify/blob/master/docs/Canvas.jpg***

2. Cronograma do Trabalho

A seguir é apresentado o cronograma proposto para as etapas deste trabalho que deve ser seguido disciplinadamente para assegurar às entregas dos devidos artefatos nas datas estipuladas pelo curso.

Datas		**Atividade / Tarefa**	**Produto / Resultado**
De	**Até**		
21 / 01 / 23	21 / 01 /23	1. Desenvolver cronograma do projeto	Cronograma documentado no relatório técnico
21 / 01 /23	23 / 01 / 23	2. Estudar a contextualização do projeto e do mercado envolvido	Descrição detalhada sobre o objetivo do projeto e sobre o mercado que está inserido documentado no relatório técnico
24/ 01/ 23	25 / 01 / 23	3. Estudar e desenvolver o modelo de negócio em formato Canvas	Modelo Canvas detalhando o modelo de negócios completo e documentado no relatório técnico
25 /01/ 23	30/ 01 / 23	4. Definição de requisitos funcionais	Lista de requisitos funcionais documentada no relatório técnico
25/ 01 / 23	25 /01 /23	5. Definição requisitos não funcionais	Lista de requisitos não funcionais documentada no relatório técnico
25/ 01/ 23	30/01/ 23	6. Elaborar diagrama de contexto	Diagrama documentado no relatório técnico
25 / 01 / 23	25/ 01 / 23	7. Elaborar diagrama de containers	Diagrama documentado no relatório técnico
25/ 01 / 23	30 /01 / 23	8. Elaborar diagrama de componentes	Diagrama documentado no relatório técnico
01/02 /23	10 / 02 / 23	9. Gravar vídeo de apresentação ETAPA 1	Vídeo enviado e anexo na plataforma Canvas
01 / 02 / 23	10 / 02/ 23	10. Construir apresentação PPT da ETAPA	Arquivo enviado e anexo na plataforma Canvas
01 / 02 / 23	10 /02 /23	11. Disponibilizar em repositório no GITHUB, artefatos produzidos na ETAPA 1	Documentos e artefatos gerados publicados em repositório GitHub e submetidos para avaliação na plataforma Canvas.
10/ 02 / 23	12 / 02 / 23	12. Estudar formas de implantação em nuvem e provedores de cloud	N/A
13 / 02 / 23	15/ 02 / 23	13. Estudar boas práticas de arquitetura em nuvem	N/A
16 / 02 / 23	18 / 02 / 23	14. Elaborar e desenhar diagrama de implantação em nuvem	Desenho da arquitetura documentado no relatório técnico
20 / 02 / 23	25 / 02/ 23	15. Elaborar e desenhar arquitetura de dados	Documentação sobre detalhamento do banco de dados e suas tabelas (ou documentos) relacionadas

01 / 03/ 23	30 / 03/ 23	16. Desenvolver API backend e implementar os requisitos funcionais principais	Código publicado em repositório no GitHub
01/ 04/ 23	30 /04 / 23	17. Desenvolver SPA frontend	Código publicado em repositório no GitHub
01 / 04 / 23	30 / 04 / 23	18. Desenvolver BFF backend	Código publicado em repositório no GitHub
01 / 05 / 23	15 / 05 / 23	19. Implementar e testar arquitetura kubernetes localmente	Código de infraestrutura publicado em repositório no GitHub
15 / 05/ 23	20 / 05 / 23	20. Publicar objetos kubernetes no GKE (Google Kubernetes Engine)	Projeto funcionando online com URL DNS ou endereço de IP disponível aos avaliadores do projeto
20 /05 / 23	30 / 05 / 23	21. Realizar análise crítica da arquitetura e dos resultados obtidos e documentar no relatório técnico	Resultados documentados no relatório técnico utilizando a metodologia ATM
01/ 06 / 23	12/ 06 / 23	22. Realizar revisão de documentos disponibilizados na Etapa 2.	Documentos submetidos na plataforma Canvas para avaliação

3. Especificação Arquitetural da solução

Esta seção apresenta a especificação básica da arquitetura da solução a ser desenvolvida, incluindo diagramas, restrições e requisitos do projeto.

3.1 Requisitos Funcionais

Abaixo se encontra todos os requisitos funcionais, que são as funcionalidades necessários visando cobrir os objetivos do projeto Appointify.

ID	Descrição Resumida	Dificuldade (B/M/A) *	Prioridade (B/M/A) *
RF01	O sistema deve permitir o auto cadastramento do usuário e seu perfil (CLIENTE, FUNCIONÁRIO, EMPRESA)	B	A
RF02	Gestão de empresas: O sistema deve permitir o cadastro de novas empresas (estabelecimentos, pequenos comércios e prestadores de serviço independentes).	B	A
RF03	Gerenciamento de funcionários: O sistema deve permitir que os administradores gerenciem os funcionários do estabelecimento (podendo eles mesmos serem funcionários), incluindo adicionar, editar e excluir barbeiros.	B	A
RF04	Gerenciamento de horários: O sistema deve permitir que os administradores gerenciem os horários disponíveis, incluindo adicionar, editar e excluir horários.	M	A
RF05	Agendamento de serviços: O sistema deve permitir que os clientes agendem serviços com base em horários disponíveis e forneçam informações de contato.	B	A
RF06	Notificações: O sistema deve enviar notificações por e-mail ou mensagem de texto para os clientes sobre confirmações de agendamento, lembretes de agendamento e cancelamentos.	M	B
RF07	Integração com outros sistemas: O sistema deve ser capaz de integrar-se com outros sistemas, como sistemas de pagamento, para facilitar o processo de agendamento.	A	B
RF08	Acesso móvel: O sistema deve ser acessível via dispositivos móveis (interface responsiva), permitindo que os clientes agendem e gerem seus serviços a partir de qualquer lugar.	A	A
RF09	Relatórios: O sistema deve fornecer relatórios sobre o desempenho dos agendamentos, incluindo estatísticas sobre o número de agendamentos, horários mais populares e funcionários mais solicitados	A	B
RF10	Segurança: O sistema deve implementar medidas de segurança para proteger as informações dos clientes e garantir a confidencialidade dos dados como criptografia de senhas.	M	A
RF11	Suporte ao cliente: O sistema deve fornecer suporte ao cliente para ajudar os clientes a resolver problemas com seus agendamentos, nessa primeira versão do aplicativo se espera uma sessão de FAQ (perguntas frequentes) e informações de contato.	B	B
RF12	Personalização: O sistema deve permitir que os administradores personalizem a aparência da página do seu estabelecimento. Na página do estabelecimento poderão alterar a foto/banner do estabelecimento bem como informações cadastras e informações de serviços oferecidos.	A	B

RF13	Gerenciamento de serviços: O sistema deve permitir que os administradores gerem serviços oferecidos pelo salão, incluindo adicionar, editar e excluir serviços.	M	A
RF14	Gerenciamento de preços: O sistema deve permitir que os administradores gerem preços dos serviços oferecidos pelo salão.	B	A
RF15	Gerenciamento de pacotes: O sistema deve permitir que os administradores criem pacotes de serviços e ofereçam descontos para pacotes.	A	B
RF16	Gerenciamento de promoções: O sistema deve permitir que os administradores gerem e anunciem promoções no site ou nas redes sociais.	A	B
RF17	Gerenciamento de clientes: O sistema deve permitir que os administradores gerem informações sobre os clientes, incluindo histórico de serviços e pagamentos.	M	B
RF18	Gerenciamento de avaliações: O sistema deve permitir que os clientes deixem avaliações sobre os serviços e os estabelecimentos.	M	M
RF19	Integração com ferramentas de pagamento: O sistema deve permitir que os clientes paguem pelos serviços através de cartão de crédito ou débito, ou através de outras formas de pagamento eletrônico.	A	M
RF20	Busca de serviços e estabelecimentos: O sistema deve permitir que os clientes busquem por diferentes tipos de serviço cadastrados na plataforma	B	A

*B=Baixa, M=Média, A=Alta.

3.2 Requisitos Não-funcionais

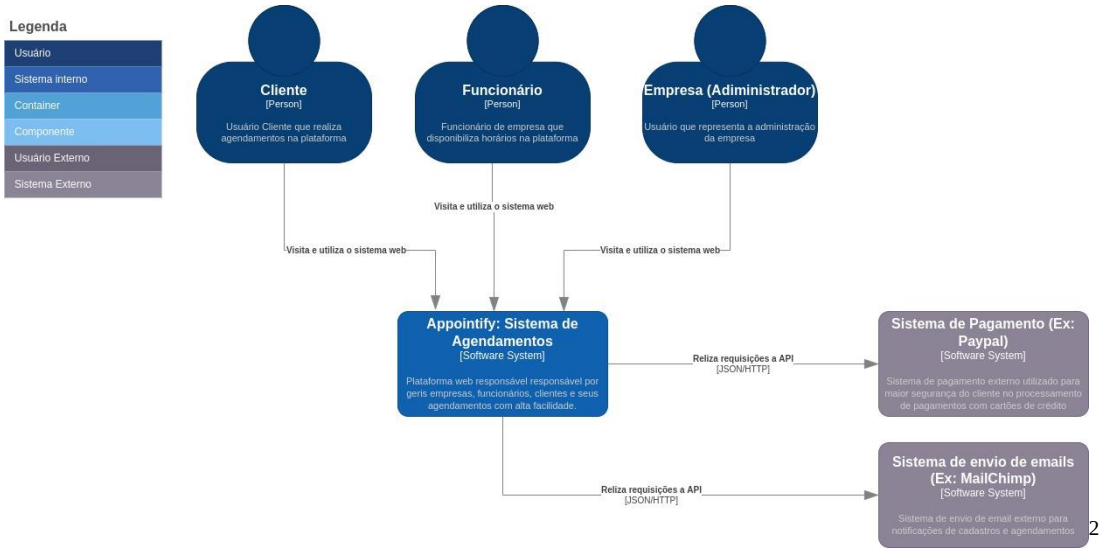
Abaixo se encontra todos os requisitos não funcionais, são aqueles que descrevem características e propriedades do sistema que não são diretamente relacionadas à funcionalidade do sistema, mas que afetam o seu desempenho, usabilidade, confiabilidade, segurança e outros aspectos importantes.

ID	Descrição	Prioridade B/M/A
RNF01	Disponibilidade: O sistema deve estar disponível para uso constantemente, sem interrupções ou falhas.	A
RNF02	Desempenho: O sistema deve responder rapidamente às solicitações dos usuários e processar informações de forma eficiente.	M
RNF03	Escalabilidade: O sistema deve ser capaz de lidar com aumentos na carga de trabalho, como um aumento no número de usuários ou agendamentos.	A
RNF04	Segurança: O sistema deve proteger as informações dos usuários contra acesso não autorizado, ataques de hackers e outras ameaças de segurança.	M
RNF05	Manutenibilidade: O sistema deve ser fácil de manter e atualizar, permitindo que os administradores adicionem novos recursos ou corrijam problemas rapidamente.	A
RNF06	Compatibilidade: O sistema deve ser compatível com diferentes tipos de dispositivos, sistemas operacionais e navegadores.	M
RNF07	Usabilidade: O sistema deve ser fácil de usar e navegar para os usuários, incluindo opções de ajuda e instruções claras.	B
RNF08	Testabilidade: O sistema deve ser facilmente testável para garantir a qualidade e a estabilidade.	A
RNF09	Integração: O sistema deve ser capaz de se integrar com outros sistemas, como sistemas de pagamento, sistemas de gerenciamento de clientes, e outras ferramentas.	A

*B=Baixa, M=Média, A=Alta.

3.3 Diagrama de Contexto

O nível de abstração 1 dos diagramas C4, consiste no diagrama de contexto, o qual representa a visão geral do sistema de software que está sendo desenvolvido e sua relação com o ambiente externo, incluindo os atores (tanto humanos quanto sistemas) que o utilizam e com os quais ele se comunica. Este diagrama é útil para comunicar aos stakeholders como o sistema será utilizado e quais funcionalidades serão fornecidas, e é geralmente o primeiro diagrama a ser desenvolvido durante o processo de análise de requisitos. Abaixo segue o diagrama de contexto do projeto Appointify que está sendo desenvolvido nesse relatório técnico.



[2]

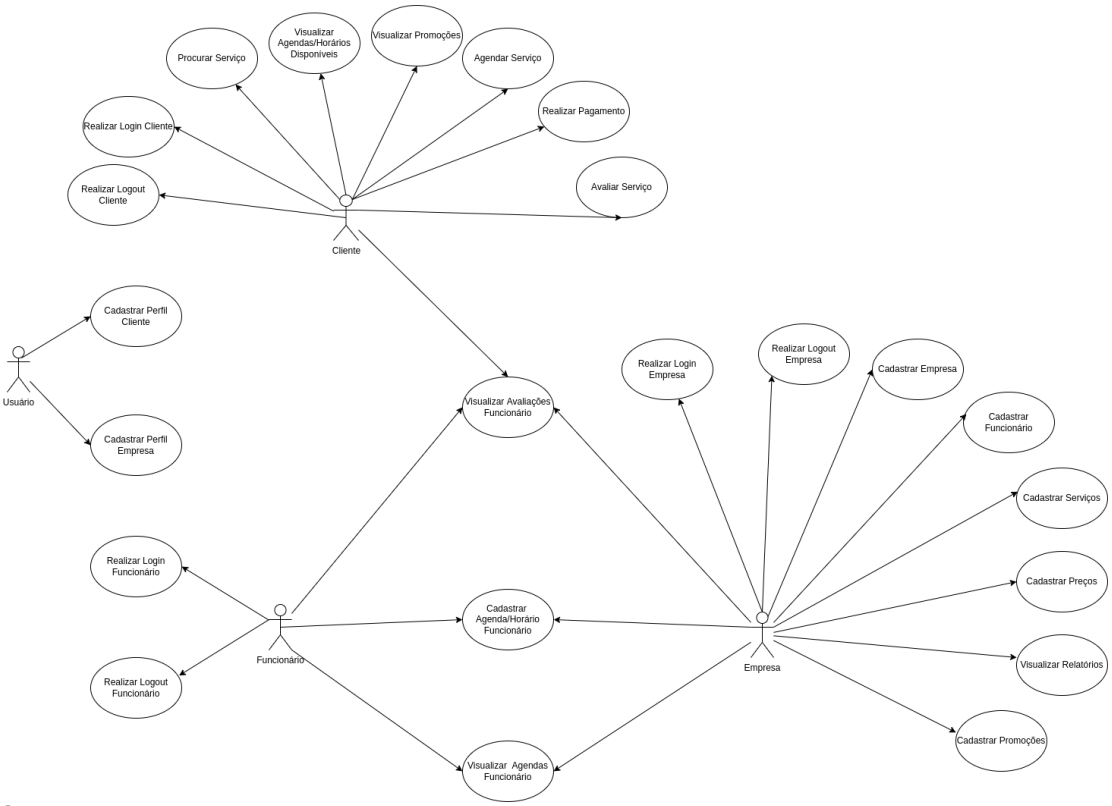
A figura acima, mostra o Diagrama de Contexto referente ao projeto Appointify, nela podemos identificar os elementos a seguir:

- Persona 1: O Usuário Cliente representa os clientes em potencial e tem acesso às funções de auto cadastro, pesquisa por estabelecimentos, agendamentos, criação de tickets de atendimento, realização de avaliações e pagamentos.
- Persona 2: O Usuário Funcionário é cadastrado por outros funcionários de nível administrador ou pelo usuário empresa e tem acesso à gestão de outros funcionários (se for administrador) e criação de horários.
- Persona 3: O Usuário Empresa é a conta de administrador responsável por cadastrar a empresa na plataforma e tem acesso às funcionalidades de cadastrar funcionários, monitorar relatórios, gestão da página do estabelecimento, cadastro de serviços e controle de preços.
- Sistema Interno: O sistema Appointify é representado por um retângulo azul no diagrama de contexto, indicando que é um sistema interno e sujeito ao nosso controle. O diagrama apresenta o sistema de forma macro, sem entrar em detalhes quanto aos seus componentes internos. No entanto, é possível observar a integração do Appointify com sistemas externos de envio de e-mails e pagamento, realizados por outras empresas.

[2] **Figura – Diagrama de Contexto. Disponível em: https://github.com/RodrigoTopan/appointify/blob/master/docs/Diagrama%20de%20Contexto.jpg**

- Sistemas Externos: Esses sistemas são representados por retângulo cinzas, sinalizando que são de propriedade de parceiros de negócio que fornecem formas de integração, como API's. Visando maior segurança e confiança aos clientes e mais facilidade no processamento de, por exemplo, pagamentos planejamos a integração com esses sistemas externos.

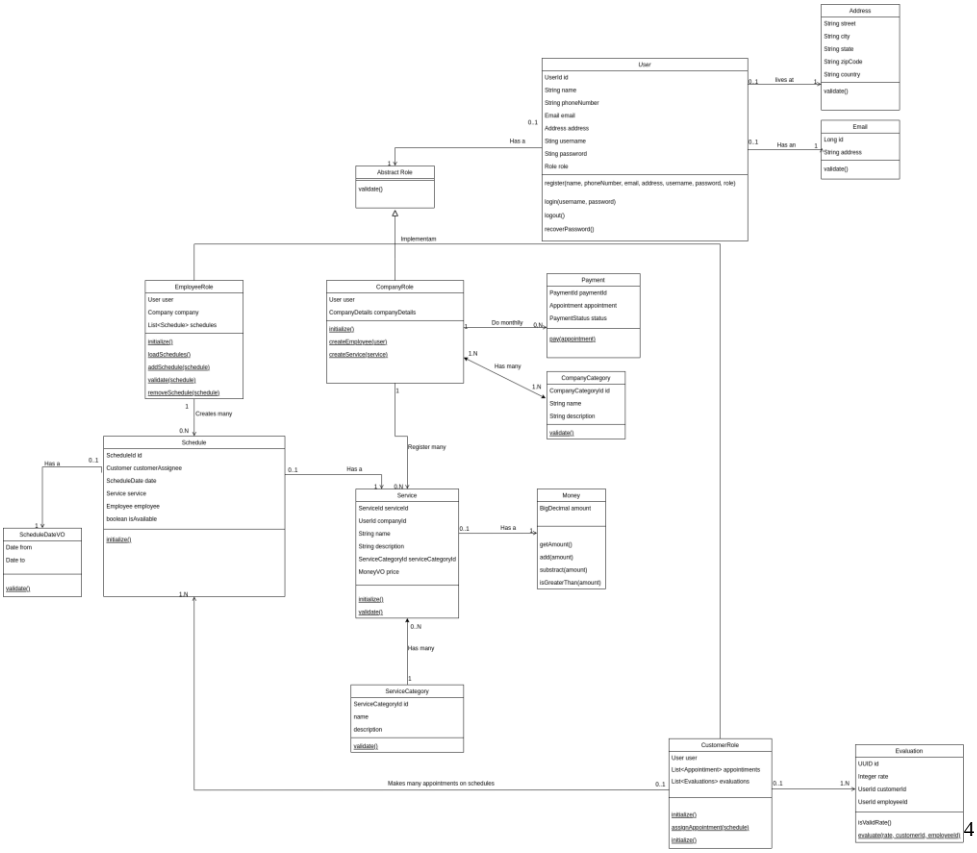
3.4 Diagrama de Casos de Uso



3

[3] **Figura – Diagrama de Casos de Uso. Disponível em: https://github.com/RodrigoTopan/appointify/blob/master/docs/diagrama-casos-uso.png**

3.5 Diagrama de Classes de Domínio



[4]

[4] **Figura – Diagrama de Classes de domínio. Disponível em: https://github.com/RodrigoTopan/appointify/blob/master/docs/Diagrama-classe-dominio.png**

4. Modelagem Arquitetural

Esta seção apresenta a modelagem arquitetural da solução proposta para o sistema Appointify, de forma a permitir seu completo entendimento. Ela descreve as principais funcionalidades e características dos usuários do sistema, bem como a integração com sistemas externos. A compreensão da modelagem arquitetural é fundamental para a implementação da prova de conceito e para o sucesso no desenvolvimento e implantação do sistema.

Para esta modelagem arquitetural optou-se por utilizar o modelo C4 para documentação de arquitetura de software.

O modelo C4 (Contexto, Container, Componente, Código) é uma metodologia de documentação de arquitetura de software que foi desenvolvida para ajudar a comunicar a arquitetura de um sistema de forma clara e consistente. Ele se concentra em modelar os diferentes níveis de abstração do sistema, permitindo que os arquitetos e desenvolvedores possam comunicar de forma eficaz a arquitetura de um sistema para diferentes stakeholders.

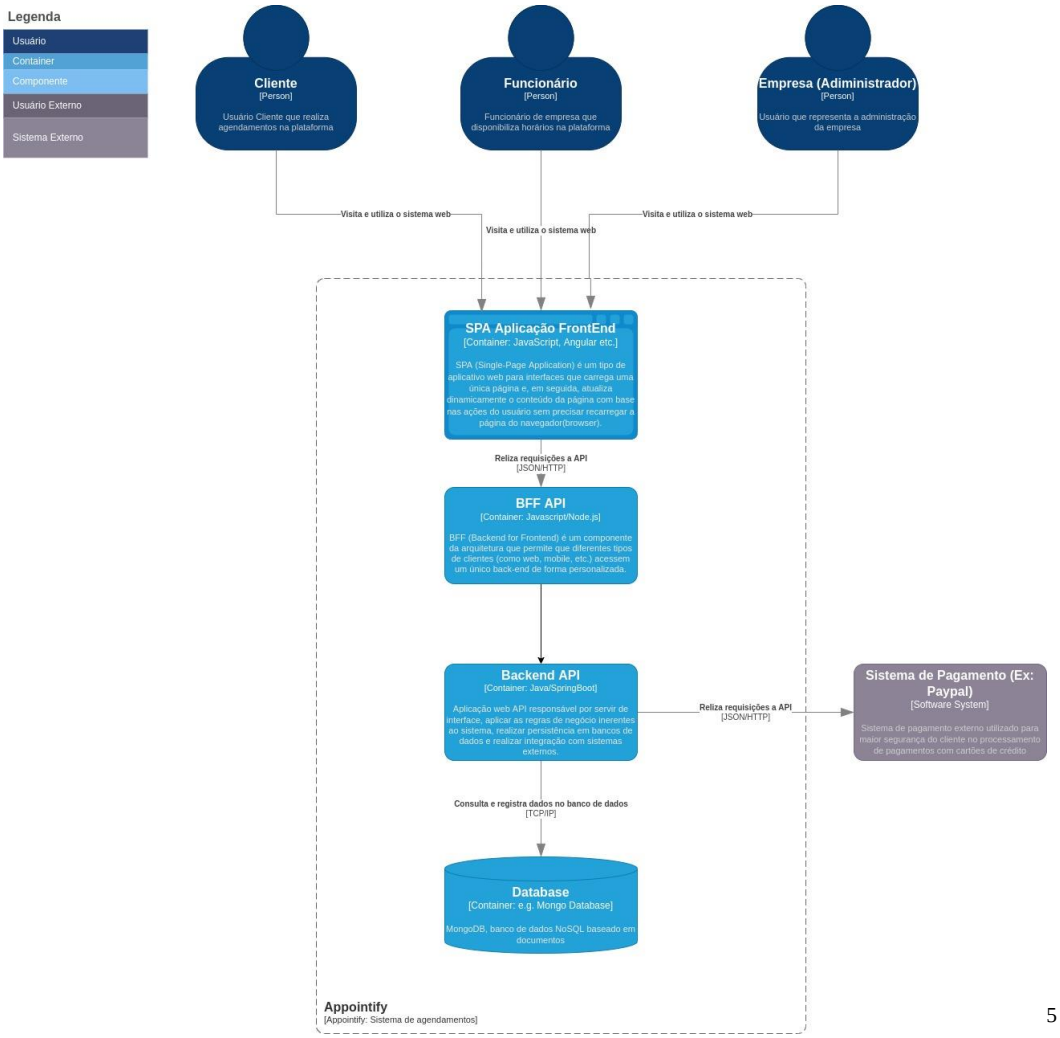
O modelo C4 é composto por quatro níveis:

- Contexto: Este nível mostra a arquitetura do sistema em um contexto maior. Ele inclui informações sobre os principais stakeholders, os principais usuários e os principais sistemas com os quais o sistema em questão se comunica.
- Containers: Este nível mostra como o sistema é dividido em contêineres (por exemplo, sistemas, processos, threads) e como esses contêineres interagem entre si.
- Componentes: Este nível mostra como os contêineres são divididos em componentes (por exemplo, classes, módulos, pacotes) e como esses componentes interagem entre si.
- Código: Este nível mostra como os componentes são implementados em código-fonte. Ele é opcional e geralmente é usado somente para desenvolvedores que trabalham no sistema.

O modelo C4 permite aos desenvolvedores e arquitetos identificarem claramente as responsabilidades de cada camada e como elas se relacionam entre si, possibilitando uma melhor compreensão do sistema e facilidade na manutenção e evolução.

Dos quatro nível que compõem o modelo C4 três serão apresentados aqui e somente o Código será apresentado nas próximas sessões do relatório técnico.

4.1 Diagrama de Container



[5]

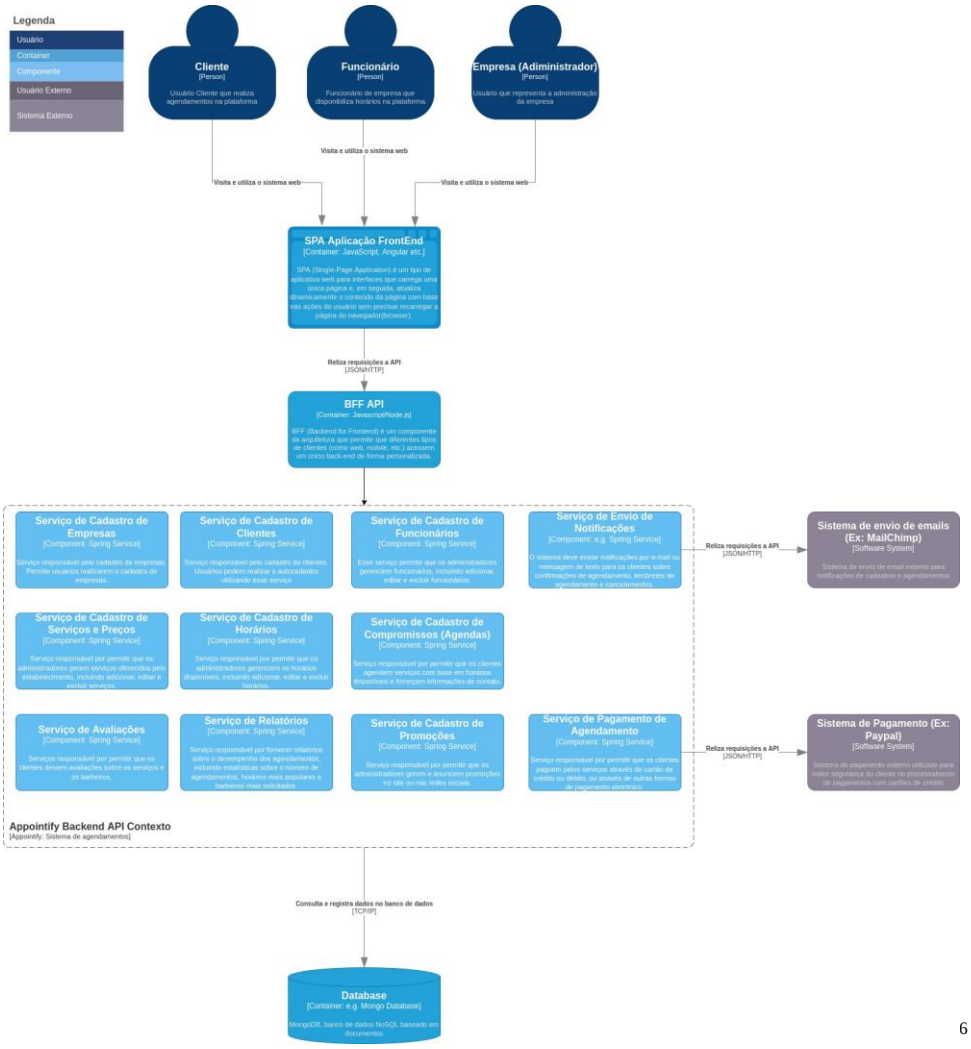
A figura acima apresenta os *containers* da aplicação, nela podemos observar mais detalhamento em relação a alguns elementos apresentados no Diagrama de Contexto, elementos a seguir:

- SPA (Single Page Applications) Frontend (Angular): Container responsável por ser a interface do usuário. SPA's são aplicações de interface gráfica que carregam apenas uma página ao usuário, dando uma sensação de maior velocidade e fluidez na exibição de conteúdo. Essa aplicação é responsável por se integrar com o BFF do Appointify

[5] **Figura – Diagrama de Container. Disponível em: https://github.com/RodrigoTopan/appointify/blob/master/docs/C4%20-%20Diagrama%20de%20Container.jpg**

- BFF (Node.js): BFF (Backend for Frontend) é um container da arquitetura que permite que diferentes tipos de clientes (como web, mobile, etc.) acessem um único backend de forma personalizada.
- Backend API (Spring): Aplicação web API responsável por aplicar as regras de negócio inerentes ao sistema, realizar persistência em bancos de dados e realizar integração com sistemas externos.
- Banco de Dados (MongoDB): Banco de dados não relacional baseado em documentos. Container responsável por armazenar todos os dados produzidos pelo Backend

4.2 Diagrama de Componentes



[6]

A figura acima apresenta os *componentes* da aplicação, nela podemos observar mais detalhamento em relação a alguns elementos apresentados no Diagrama de Containers, elementos a seguir:

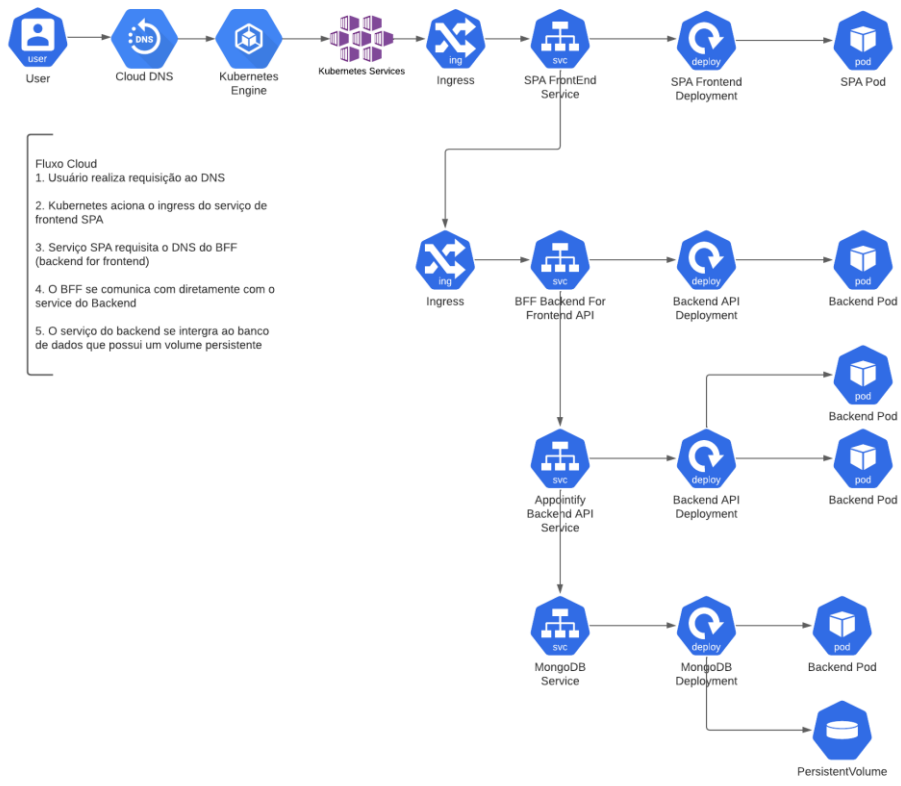
- Componente de Cadastro de Empresas (Serviço em Spring): Serviço responsável pelo cadastro de empresas. Permite usuários realizarem o cadastro de empresas.
- Componente de Cadastro de Clientes (Serviço em Spring): Serviço responsável pelo cadastro de clientes. Usuários podem realizar o auto cadastro utilizando esse serviço

[6] **Figura – Diagrama de Componentes. Disponível em: https://github.com/RodrigoTopan/appointify/blob/master/docs/Diagrama%20de%20Componentes%20C4%20-%20Com%20m%C3%B3dulos.jpg**

- Componente de Cadastro de Funcionários (Serviço em Spring): Esse serviço permite que os administradores gerenciem funcionários, incluindo adicionar, editar e excluir funcionários.
- Componente de Envio de Notificações (Serviço em Spring): O sistema deve enviar notificações por e-mail ou mensagem de texto para os clientes sobre confirmações de agendamento, lembretes de agendamento e cancelamentos.
- Componente de Cadastro de Serviços e Preços (Serviço em Spring): Serviço responsável por permitir que os administradores gerem serviços oferecidos pelo estabelecimento, incluindo adicionar, editar e excluir serviços.
- Componente de Cadastro de Horários (Serviço em Spring): Serviço responsável por permitir que os administradores gerenciem os horários disponíveis, incluindo adicionar, editar e excluir horários.
- Componente de Cadastro de Compromissos/Agendas (Serviço em Spring): Serviço responsável por permitir que os clientes agendem serviços com base em horários disponíveis e forneçam informações de contato.
- Componente de Avaliações (Serviço em Spring): Serviços responsável por permitir que os clientes deixem avaliações sobre os serviços e os barbeiros.
- Componente de Relatórios (Serviço em Spring): Serviço responsável por gerar relatórios sobre o desempenho dos agendamentos, incluindo estatísticas sobre o número de agendamentos, horários mais populares e barbeiros mais solicitados.
- Componente de Cadastro de Promoções (Serviço em Spring): Serviço responsável por permitir que os administradores gerem e anunciem promoções no site ou nas redes sociais.
- Componente de Pagamento de Agendamento (Serviço em Spring): Serviço responsável por permitir que os clientes paguem pelos serviços através de cartão de crédito ou débito, ou através de outras formas de pagamento eletrônico.

5. *Arquitetura de implantação em nuvem*

A seguir, desenho da implantação da arquitetura em nuvem, listando todos os recursos necessários para o funcionamento online na Google Cloud Platform (Plataforma Cloud do Google). Nesta arquitetura, visando ser agnósticos quanto a cloud, optamos pelo uso do Kubernetes, uma plataforma de orquestração de contêiners que pode ser usado em qualquer provedor cloud. Dessa forma, se optarmos por trocar de provedor no futuro, o processo de migração é facilitado.



[7]

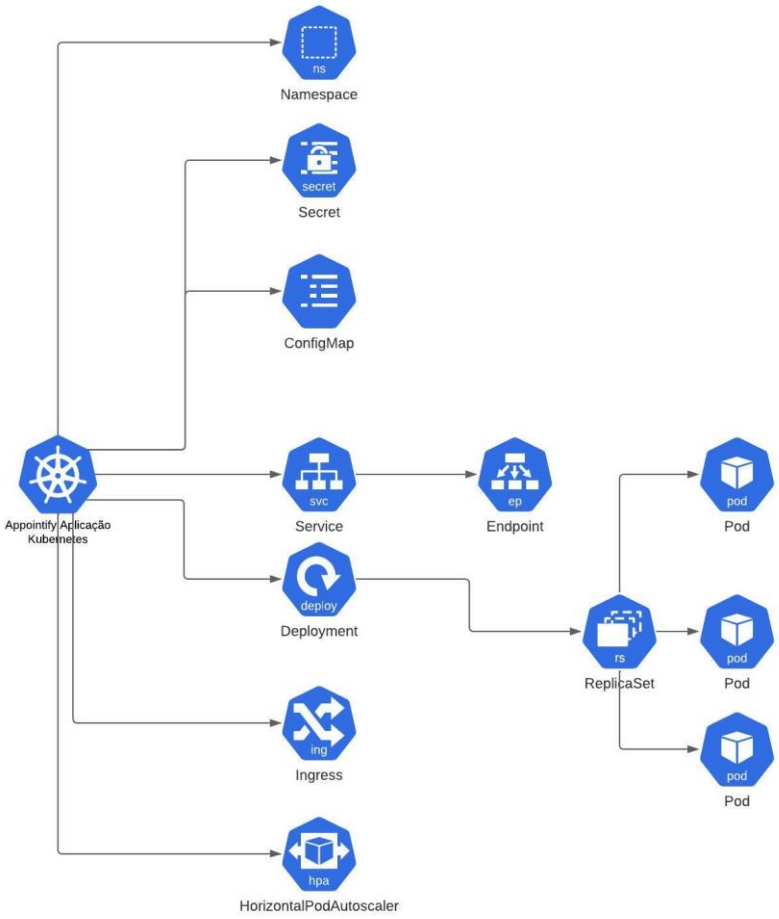
O diagrama acima explica todos os componentes cloud utilizados. Na prática são dois componentes da Google Cloud necessários para rodar essa arquitetura: o gerenciador de DNS e a Engine do Kubernetes.
O Fluxo Cloud descrito acima, consiste em:

- Usuário realiza requisição ao DNS Kubernetes aciona o componente de ingresso do serviço de frontend SPA.
- Serviço SPA requisita o DNS do BFF (backend for frontend) e se integra a essa API.

[7] **Figura – Diagrama de implantação em nuvem.**

Disponível em: https://github.com/RodrigoTopan/appointify/blob/master/docs/Kubernetes-Fluxo.png

- O BFF se comunica diretamente com o serviço do Backend via FQDN (Fully Qualified Domain Name, ou em Português, Nome de Domínio Completamente Qualificado)
- O serviço do backend se integra ao banco de dados que possui um volume persistente e mantem os dados salvos e seguros.



[8]

O diagrama acima explica como cada objeto dessa arquitetura kubernetes se integra, exemplificando com mais detalhes o que foi apresentado no diagrama de integrações anterior. Segue uma breve explicação do que são cada um destes objetos kubernetes utilizados dentro da Google Cloud Kubernetes Engine.

[8] **Figura – Diagrama de Implantação em Nuvem com Objetos Kubernetes Detalhados**
Disponível em: https://github.com/RodrigoTopan/appointify/blob/master/docs/Arquitetura-de-nuvem.png

- Namespace: Um namespace é um espaço utilizado para dividir recursos em um cluster Kubernetes. Ele é usado para isolamento e organização de recursos em um cluster.
- Secret: Uma Secret é um objeto do Kubernetes que armazena informações confidenciais, como senhas, tokens e chaves. Ele é criptografado e pode ser usado para autenticação e autorização de acesso a recursos sensíveis, como banco de dados.
- ConfigMap: Um ConfigMap é um objeto do Kubernetes que armazena dados de configuração em formato de chave-valor. Ele pode ser usado para fornecer configurações para pods e outros recursos do Kubernetes. Esse recurso geralmente é utilizado para fornecer varíaveis de ambiente aos pods.
- Ingress: Um Ingress é um recurso do Kubernetes que controla o tráfego de entrada para um cluster. Ele pode ser usado para mapear endereços externos para serviços internos e para aplicar regras de roteamento e autenticação.
- Service: Um Service é um recurso do Kubernetes que fornece um ponto de acesso único para um conjunto de pods. Ele é usado para garantir a disponibilidade e escalabilidade de aplicações. Um pod não se comunica diretamente com outro pod por estes serem efêmeros. Para isso utilizamos os services que realizam o roteamento e balanceamento de carga entre pods.
- Deployment: Um Deployment é um objeto do Kubernetes que gerencia a implantação de aplicações. Utilizamos o deployment para atualizar a versão do aplicativo e garantir que sempre essa versão esteja em execução.
- ReplicaSet: Um ReplicaSet é um recurso do Kubernetes que gerencia o escalonamento de pods. Ele é usado para garantir que um determinado número de réplicas de um pod estejam sempre em execução.
- Pod: Um pod é a unidade básica de execução em um cluster Kubernetes. Ele é usado para executar um ou mais contêineres. Dentro dos pods está em execução o código da nossa aplicação em uma pequena máquina virtual chamda (Contêiner)
- Persistent Volume: Um Persistent Volume é um recurso do Kubernetes que fornece armazenamento persistente para pods. Ele é usado para garantir que os dados de uma aplicação sejam preservados mesmo quando os pods são

recriados. Utilizamos para persistir os dados geridos pelo SGBD (Sistema gerenciador de banco de dados). No caso, o MongoDB.

6. Avaliação da Arquitetura (ATAM)

A avaliação da arquitetura desenvolvida neste projeto é abordada nesta seção visando avaliar se ela atende ao que foi solicitado pelo cliente, segundo o método ATAM.
O método ATAM (Architecture Tradeoff Analysis Method) é uma metodologia para avaliar a qualidade de uma arquitetura de software. Ela inclui uma série de etapas para avaliar diferentes aspectos da arquitetura, incluindo requisitos, restrições, riscos e outros fatores.

6.1. Análise das abordagens arquiteturais

Abaixo, resumo das principais características da proposta arquitetural utilizando o método Architecture Tradeoff Analysis Method (ATAM), no qual são utilizados cenários para fazer essa análise.

Atributos de Qualidade	Cenários	Importância	Complexidade
Interoperabilidade	O sistema deve se comunicar com sistemas de outras tecnologias.	A	M
Usabilidade	O sistema deve prover boa usabilidade.	M	B
Manutenibilidade	O sistema deve ser fácil de manter e atualizar.	M	M
Compatibilidade	O sistema deve ser compatível com diferentes dispositivos e sistemas operacionais.	M	M
Requisitos	O sistema deve atender aos requisitos dos clientes e administradores, incluindo a capacidade de agendar serviços, gerenciar horários, enviar notificações e gerenciar funcionários.	A	A
Restrições	O sistema deve atender a restrições legais e regulamentares, incluindo as normas de proteção de dados.	A	B
Riscos	O sistema deve mitigar riscos, como falhas de segurança ou problemas de disponibilidade.	A	M

6.2. Resultados Obtidos

Abaixo, uma tabela explicando os resultados da arquitetura produzida, indicando seus pontos fortes e suas limitações.

Requisitos Não Funcionais	Teste	Homologação
RNF01: O sistema deve estar disponível para uso constantemente, sem interrupções ou falhas.	OK	OK
RNF02: O sistema deve responder rapidamente às solicitações dos usuários e processar informações de forma eficiente.	OK	OK
RNF03: O sistema deve ser capaz de lidar com aumentos na carga de trabalho, como um aumento no número de usuários ou agendamentos.	OK	OK
RNF04: O sistema deve proteger as informações dos usuários contra acesso não autorizado, ataques de hackers e outras ameaças de segurança.	OK	OK
RNF05: O sistema deve ser fácil de manter e atualizar, permitindo que os administradores adicionem novos recursos ou corrijam problemas rapidamente.	OK	OK
RNF06: O sistema deve ser compatível com diferentes tipos de dispositivos, sistemas operacionais e navegadores.	OK	OK
RNF07: O sistema deve ser fácil de usar e navegar para os usuários, incluindo opções de ajuda e instruções claras.	OK	N/A
RNF08: O sistema deve ser facilmente testável para garantir a qualidade e a estabilidade.	OK	N/A
RNF09: O sistema deve ser capaz de se integrar com outros sistemas, como sistemas de pagamento, sistemas de gerenciamento de clientes, e outras ferramentas.	OK	OK

Segue uma breve explicação de como cada um dos requisitos não funcionais do projeto foram atendidos:

- **RNF01:** O sistema foi implantado em nuvem pública (Google Cloud) com kubernetes o que permite ao sistema alta disponibilidade usando os recursos computacionais do provedor e objetos kubernetes como load balancer para distribuição de carga e o HPA para garantir que a aplicação esteja em execução e disponível.
- **RNF02:** A API do sistema foi desenvolvida com Java Spring Boot o que garante boa eficiência na execução dos processos da aplicação. Também foi utilizado melhores técnicas no desenvolvimento do código para redução de sua complexidade de execução como o Big O.
- **RNF03:** O sistema foi desenvolvido utilizando o alto poder computacional da núvem do Google (Google Cloud) permitindo escalabilidade horizontal conforme demanda.

- **RNF04:** O sistema implementa métodos de autenticação e autorização para o acesso de cada usuário e controle de perfis. As informações sensíveis do usuário como senha são armazenadas criptografas no banco de dados e a autenticação é feita via JWT (Json Web Token).
- **RNF05:** O sistema foi desenvolvido baseado na "Arquitetura Limpa", conceito de arquitetura de software em camadas apresentado por Robert C.Martin e largamente difundido na indústria de desenvolvimento do software que permite o isolamento das regras de negócio "domínio" dos detalhes técnológicos como banco de dados. Permitindo assim alto desacoplamento e manutenibilidade entre as camadas do sistema.
- **RNF06:** A interface do usuário foi desenvolvida como SPA (Single Page Application) responsiva. Ou seja, o cliente pode acessar a aplicação por qualquer navegador web (browser), em qualquer dispositivo desktop e mobile.
- **RNF07:** Pensando na maior facilidade de uso do usuário e uma melhor experiência. O projeto foi desenvolvido de forma minimalista. Contendo poucas telas e o acesso aos recursos de forma rápida.
- **RNF08:** O sistema foi desenvolvido usando TDD (Test Driven Development), o sistema atingiu mais de 80% de cobertura de testes automatizados. Sendo estes testes unitários e de integração.
- **RNF09:** Dentro do system design do sistema temos o diretório de gateways que fazendo integração com fontes de dados externas como banco de dados e API's. Utilizando a biblioteca java chamada OpenFeign a integração entre API's é facilitada.

7. Avaliação Crítica dos Resultados

Abaixo tabela explicando os pontos de principais tradeoffs do projeto

Ponto avaliado	Descrição
Escalabilidade e desempenho	A arquitetura foi projetada para escalar horizontalmente, mas isso pode afetar o desempenho, portanto foi necessário encontrar um equilíbrio entre ambos para garantir uma solução eficiente.
Segurança X Facilidade de uso	A segurança é crucial, mas a facilidade de uso também foi considerada importante para garantir uma boa experiência para os usuários finais.
Flexibilidade X Complexidade	A arquitetura foi projetada para ser flexível o suficiente para lidar com mudanças futuras, mas isso pode aumentar a complexidade do projeto. No caso da adoção de arquitetura limpa, foi observada maior complexidade do que as implementações mais comuns como MVC, o que afetou a produtividade de entrega.
Manutenibilidade X Produtividade	A manutenção foi considerada importante para garantir a disponibilidade e a confiabilidade da solução, mas isso pode ter afetado a produtividade da equipe. No entanto, a adoção de arquitetura limpa possibilitou uma maior facilidade de adaptação a mudanças de infraestrutura.
Integração X Independência	A integração com outros sistemas foi considerada importante, mas isso pode afetar a independência e a flexibilidade da solução. A adoção de arquitetura limpa permitiu lidar com componentes externos de forma simplificada, protegendo o core da aplicação de alterações indesejadas.

8. Conclusão

Em conclusão, o desenvolvimento do Appointify representa uma contribuição significativa à democratização do acesso à aplicativos de agendamento de serviços para pequenos comércios e prestadores de serviço. A plataforma funciona como um marketplace, conectando estabelecimentos e prestadores de serviços a novos clientes, além de fidelizar a base de clientes existente, oferecendo facilidades para encontrar estabelecimentos bem avaliados, comparar preços e usufruir de programas de fidelidade e descontos. Para os estabelecimentos, a plataforma proporciona maior facilidade na gestão de suas atividades e visibilidade para alavancar suas vendas.

O desenvolvimento do sistema foi um projeto desafiador que envolveu todo o ciclo de desenvolvimento de software, incluindo planejamento, desenvolvimento, testes e implantação em nuvem. O objetivo principal foi fornecer uma solução eficiente e fácil de usar para os pequenos comércios, permitindo que eles gerenciem seus agendamentos de forma eficaz e aumentem sua capacidade de atendimento de forma acessível.

O modelo SaaS e a implantação em nuvem foram uma escolha estratégica para garantir segurança, escalabilidade e disponibilidade. Apesar dos desafios relacionados à segurança, desempenho e escalabilidade durante a implantação em nuvem, os esforços adicionais para garantir a segurança dos dados e a disponibilidade do sistema, bem como a implantação de estratégias de escalabilidade com Kubernetes, foram cruciais para o sucesso do projeto.

Em geral, o Appointify foi bem-sucedido em atender às necessidades e requisitos estabelecidos, proporcionando uma solução eficiente e fácil de usar para o público alvo. O Appointify é uma excelente opção para pequenos comércios e prestadores de serviço que desejam gerenciar seus agendamentos de forma eficaz e alcançar novos clientes.

Possíveis melhorias, ajustes e funcionalidades novas:

1. Calendário compartilhado: O sistema poderia permitir no futuro aos usuários compartilhamento de seus calendários com outros usuários, como colegas de trabalho ou clientes, para facilitar a coordenação de agendamentos.
2. Agendamentos recorrentes: O sistema poderia sugerir programar agendamentos com recorrência e enviar confirmações automáticas de agendamentos para os usuários, para garantir que eles tenham uma cópia do seu agendamento.

3. Login com mídias sociais (Conta google): O sistema poderia permitir o cadastro e login utilizando contas de redes sociais como google e facebook para facilitar o acesso mais rápido do usuário ao sistema
4. Desenvolver aplicativo móvel: Dessa forma o usuário poderia acessar funcionalidades nativas de dispositivos móveis, além de maior desempenho do aplicativo se comparado a um website SPA responsivo.
5. Gerenciamento de cancelamentos: O sistema poderia permitir que os usuários cancelem ou reagendem seus agendamentos, e fornecer opções para lidar com cancelamentos de última hora como estorno e reembolso de valores.
1. Geração de recibos: O sistema poderia permitir no futuro a geração de recibos para os clientes não dependerem de relatórios de agendamento do estabelecimento
2. Gerenciamento de filas: O sistema poderia permitir no futuro que aos administradores gerarem filas de agendamentos, permitindo que os usuários aguardem em lista de espera para agendamentos específicos e sejam notificados quando surgir uma vaga.
3. Gerenciamento de atendimento ao cliente: O sistema poderia permitir que os administradores criarem/abrirem tickets de atendimento ao cliente relacionadas aos agendamentos e forneçam respostas aos usuários.
4. Personalização: O sistema poderia permitir a criação de páginas mais personalizáveis aos estabelecimentos podendo se adequar melhor às necessidades das suas identidades visuais.

Lições aprendidas e constatações:

1. A necessidade de compreender completamente os requisitos do cliente antes de começar o desenvolvimento.
2. A necessidade de planejamento e gerenciamento de projetos para garantir que o projeto seja concluído dentro do prazo e orçamento.
3. A importância de testes e validação para garantir a qualidade e a confiabilidade do software.
4. A necessidade de flexibilidade e adaptabilidade para lidar com mudanças inesperadas no projeto.
5. A importância de manter a segurança e a privacidade de dados.

6. A necessidade de manter o software atualizado e escalável para atender às necessidades futuras.
7. A importância das nuvens públicas, entregando IAAS (Infraestrutura como Serviço) a todo tipo de negócio. Permitindo o acesso e desenvolvimento de soluções de software de alta disponibilidade e desempenho à um preço acessível. E Permitindo que pequenos times possam focar no negócio e não nas configurações de infraestrutura.
8. A importância da adoção da cultura devops para garantir a satisfação do cliente, entregas rápidas e a confiabilidade do software.