PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

PUC Minas Virtual

Pós-graduação *Lato Sensu* em Arquitetura de Soluções

Projeto Integrado

Relatório Técnico

MedAgenda

CLEBER RIBEIRO CAMPOS

Belo Horizonte
Novembro de 2024.

Sumário

Sumário

1. Introdução

A crescente digitalização de processos nas áreas de saúde e a demanda por sistemas integrados têm evidenciado a importância de soluções tecnológicas que otimizem a gestão clínica, principalmente em clínicas de pequeno e médio porte. No Brasil, o mercado de saúde digital cresce anualmente, destacando-se pela adoção de tecnologias de ponta que visam a automatização de serviços e o aumento da eficiência operacional. Segundo [referência de pesquisa recente], o uso de tecnologias móveis para gerenciamento clínico reduz significativamente o tempo de atendimento e melhora a experiência do paciente. Esse contexto mostra a necessidade de um sistema robusto que integre e automatize atividades essenciais, como cadastro de profissionais e pacientes e o agendamento de consultas.

O problema que este projeto busca resolver é a ausência de uma plataforma única e integrada para o gerenciamento clínico de clínicas de menor porte, que atualmente operam com processos manuais ou com ferramentas fragmentadas, limitadas e mais propensas a erros. Esses problemas dificultam o monitoramento de dados de médicos e pacientes, tornando o controle de agendamentos menos eficiente e impactando negativamente a qualidade do atendimento e a organização interna. A solução proposta visa proporcionar uma ferramenta centralizada, ágil e confiável para simplificar e otimizar esses fluxos de trabalho.

A motivação para o desenvolvimento desta solução está fundamentada em uma demanda mercadológica crescente por plataformas que tragam automação e organização para o setor de saúde. Através de um sistema integrado e móvel, espera-se alcançar benefícios como a diminuição de retrabalhos, a melhoria na gestão de dados e o aumento da satisfação dos pacientes e dos profissionais de saúde. Além disso, estudos indicam que a implantação de tecnologias em nuvem, como a AWS, pode reduzir custos operacionais e aumentar a agilidade no acesso a informações clínicas, o que reforça a importância de se investir em uma solução inovadora e acessível para clínicas de menor porte.

MedAgenda

O objetivo deste projeto é desenvolver uma solução tecnológica móvel para a clínica médica fictícia MED-CONTROL S/A, que integra o cadastro de médicos e pacientes, o agendamento e o cancelamento de consultas, além de oferecer monitoramento e gestão eficientes de dados clínicos. Esse sistema visa a alta disponibilidade, escalabilidade e segurança por meio da infraestrutura da AWS.

Os objetivos específicos propostos são:

- Realizar um estudo de mercado sobre as soluções tecnológicas voltadas ao gerenciamento de clínicas de pequeno e médio porte;
- Descrever de forma detalhada os requisitos funcionais e não funcionais da aplicação;
- Desenvolver protótipos de interface e diagramas dos principais casos de uso, considerando aspectos de experiência do usuário;
- Definir e documentar o padrão arquitetural da solução com integração à AWS, visando garantir alta disponibilidade e segurança;
- Implementar um plano de testes e validações para assegurar a usabilidade e confiabilidade do sistema.

Esta proposta de valor almeja oferecer uma plataforma intuitiva e eficiente que elimine processos manuais, garantindo uma melhor gestão e otimização do tempo dos profissionais de saúde, ao mesmo tempo em que melhora a qualidade do atendimento aos pacientes.

2. Cronograma do Trabalho

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

Datas		**Atividade / Tarefa**	**Produto / Resultado**
De	**Até**		
5/10/2024	15/10/2024	1. Revisar materiais e requisitos do projeto.	Entendimento geral do projeto.
16/10/2024	17/10/2024	2. Relatório Técnico: Definir objetivos do projeto, apresentar o problema e descrever o escopo do sistema.	Confecção do Relatório Técnico – Descritivos.
18/10/2024	19/10/2024	3. Relatório Técnico: Identificar atores, requisitos funcionais e não funcionais.	Confecção do Relatório Técnico – Definição conceitual.
20/10/2024	24/10/2024	4. Criação do protótipo de interface e definição dos elementos de navegação.	Confecção do Relatório Técnico – Inclusão das referências do Protótipo.
25/10/2024	27/10/2024	5. Relatório Técnico: Desenvolver o Diagrama de Classes de Domínio.	Confecção do Relatório Técnico – Diagrama de Classes de Domínio.
28/10/2024	29/10/2024	6. Relatório Técnico: Definir o padrão arquitetural e as tecnologias utilizadas.	Confecção do Relatório Técnico – Descrição da Arquitetura escolhida e tecnologias utilizadas.
30/10/2024	31/10/2024	7. Relatório Técnico: Criar o Diagrama de Contexto do Projeto e explicação do C4 Model.	Confecção do Relatório Técnico – Diagrama de Contexto (C4 Model) e explicação.
01/11/2024	02/11/2024	8. Relatório Técnico: Apresentação dos frameworks e estrutura do front-end.	Confecção do Relatório Técnico – Descrição dos Frameworks e visualização do layout do front end.
03/11/2024	04/11/2024	9. Relatório Técnico: Desenvolvimento do Modelo Relacional do Banco de Dados.	Confecção do Relatório Técnico – Diagrama do Modelo Relacional.
05/11/2024	05/11/2024	10. Relatório Técnico: Criação do Plano de Testes e apropriação de horas.	Confecção do Relatório Técnico – Descrição dos Testes e Apropriação de Horas.
06/11/2024	06/11/2024	11. Finalização do projeto com descrições finais e retrospectiva.	Confecção do Relatório Técnico – Descrição Finais como Retrospectiva, Objetivos Estimados e Alcançados

Tabela 1 – Cronograma de Trabalho

3. Especificação Arquitetural da solução

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

3.1 Requisitos Funcionais

Essa tabela reflete os requisitos funcionais do sistema, priorizando funcionalidades essenciais para garantir a operação básica, com foco em alta disponibilidade, segurança e escalabilidade.

ID	Descrição Resumida	Dificuldade (B/M/A)*	Prioridade (B/M/A)*
RF01	O sistema deve permitir o auto cadastramento do usuário (paciente) no aplicativo móvel.	Baixa	Alta
RF02	O sistema deve permitir o cadastro completo de médicos, incluindo nome, e-mail, telefone, CRM, especialidade e endereço.	Média	Alta
RF03	O sistema deve permitir o cadastro completo de pacientes, incluindo nome, e-mail, telefone, CPF e endereço.	Média	Alta
RF04	O sistema deve permitir a edição dos dados de médicos e pacientes, com restrições nos campos e-mail, CPF e CRM.	Média	Alta
RF05	O sistema deve permitir a listagem paginada de médicos e pacientes, com ordenação alfabética por nome.	Baixa	Média
RF06	O sistema deve permitir a inativação lógica de médicos e pacientes.	Baixa	Alta
RF07	O sistema deve permitir o agendamento de consultas com escolha do médico, paciente e data/hora, seguindo regras definidas.	Alta	Alta
RF08	O sistema deve verificar a disponibilidade de médicos no momento do agendamento e bloquear horários já ocupados.	Alta	Alta
RF09	O sistema deve selecionar um médico aleatoriamente, caso o paciente não especifique um no momento do agendamento.	Média	Média
RF10	O sistema deve enviar notificações push para lembretes de consultas e alterações no agendamento.	Alta	Alta
RF11	O sistema deve permitir o cancelamento de consultas com a especificação do motivo e um período mínimo de 24 horas.	Média	Alta

RF12	O sistema deve armazenar informações de cancelamento para fins de auditoria e relatórios.	Média	Média
RF13	O sistema deve enviar um aviso de agendamento com um dia de antecedência para pacientes e médicos.	Alta	Alta
RF14	O sistema deve permitir aos usuários visualizar e filtrar consultas agendadas, canceladas e passadas no aplicativo.	Média	Baixa
RF15	O sistema deve garantir a segurança dos dados e a conformidade com LGPD, especialmente na manipulação de dados pessoais.	Alta	Alta
RF16	O sistema deve integrar-se com serviços de nuvem AWS para escalabilidade e armazenamento seguro de dados.	Alta	Alta
RF17	O sistema deve possibilitar a geração de relatórios detalhados de consultas agendadas, canceladas e frequências.	Média	Baixa
RF18	O sistema deve permitir notificações e atualizações automáticas via AWS para garantir a disponibilidade contínua.	Alta	Baixa
RF19	O sistema deve oferecer suporte técnico via chat diretamente no aplicativo.	Média	Baixa
RF20	O sistema deve integrar-se com provedores de pagamento online para possibilitar a cobrança de taxas por transação.	Alta	Média

Tabela 2 – Requisitos Funcionais

3.2 Requisitos Não-funcionais

Esses requisitos estão estruturados para assegurar um sistema robusto, seguro, e escalável, com alta disponibilidade e desempenho adequado para o ambiente AWS. Eles oferecem suporte à confiabilidade e compatibilidade, aspectos essenciais para um sistema de agendamento médico que atende a múltiplos usuários.

ID	Descrição	Prioridade B/M/A
RNF01	O sistema deve apresentar disponibilidade 24 x 7 x 365, suportando consultas e atualizações a qualquer momento, para atender à demanda dos usuários a qualquer hora.	Alta
RNF02	O sistema deve garantir tempo de resposta de até 2 segundos para operações de busca e consulta, proporcionando uma experiência fluida para o usuário.	Alta

RNF03	A aplicação deve ser escalável horizontalmente na AWS, com recursos de aumento automático para atender a picos de acesso, como em períodos de alta demanda.	Alta
RNF04	O sistema deve utilizar mecanismos de segurança avançados, incluindo autenticação multifator (MFA) para acesso administrativo e criptografia para dados sensíveis.	Alta
RNF05	A aplicação deve estar em conformidade com a LGPD, garantindo que dados pessoais (como CPF e histórico de consultas) sejam tratados de forma segura e controlada.	Alta
RNF06	O sistema deve suportar tolerância a falhas e recuperação automática, permitindo o restabelecimento de operações em caso de falhas nos servidores ou em componentes críticos da infraestrutura.	Alta
RNF07	O sistema deve permitir a notificação push em tempo real, com envio de lembretes e atualizações de consultas aos usuários em até 5 segundos após a ação.	Média
RNF08	O aplicativo deve ter uma interface amigável e acessível, com layout responsivo e usabilidade otimizada para médicos e pacientes.	Média
RNF09	A infraestrutura deve suportar backup automático diário de dados críticos, com retenção de até 30 dias para garantir recuperação em caso de falha.	Média
RNF10	O sistema deve operar com baixo consumo de recursos no dispositivo do usuário, minimizando o uso de memória e processamento para garantir um desempenho otimizado.	Baixa
RNF11	O sistema deve ser compatível com as versões recentes dos sistemas operacionais iOS e Android (últimos 2 anos), garantindo ampla compatibilidade.	Baixa
RNF12	Os logs de atividades devem ser armazenados por pelo menos 1 ano, permitindo auditoria de acessos e ações no sistema, com facilidade de consulta.	Baixa
RNF01	O sistema deve apresentar disponibilidade 24 x 7 x 365, suportando consultas e atualizações a qualquer momento, para atender à demanda dos usuários a qualquer hora.	Alta

Tabela 3 – Requisitos Não Funcionais

3.3 Diagrama de Contexto

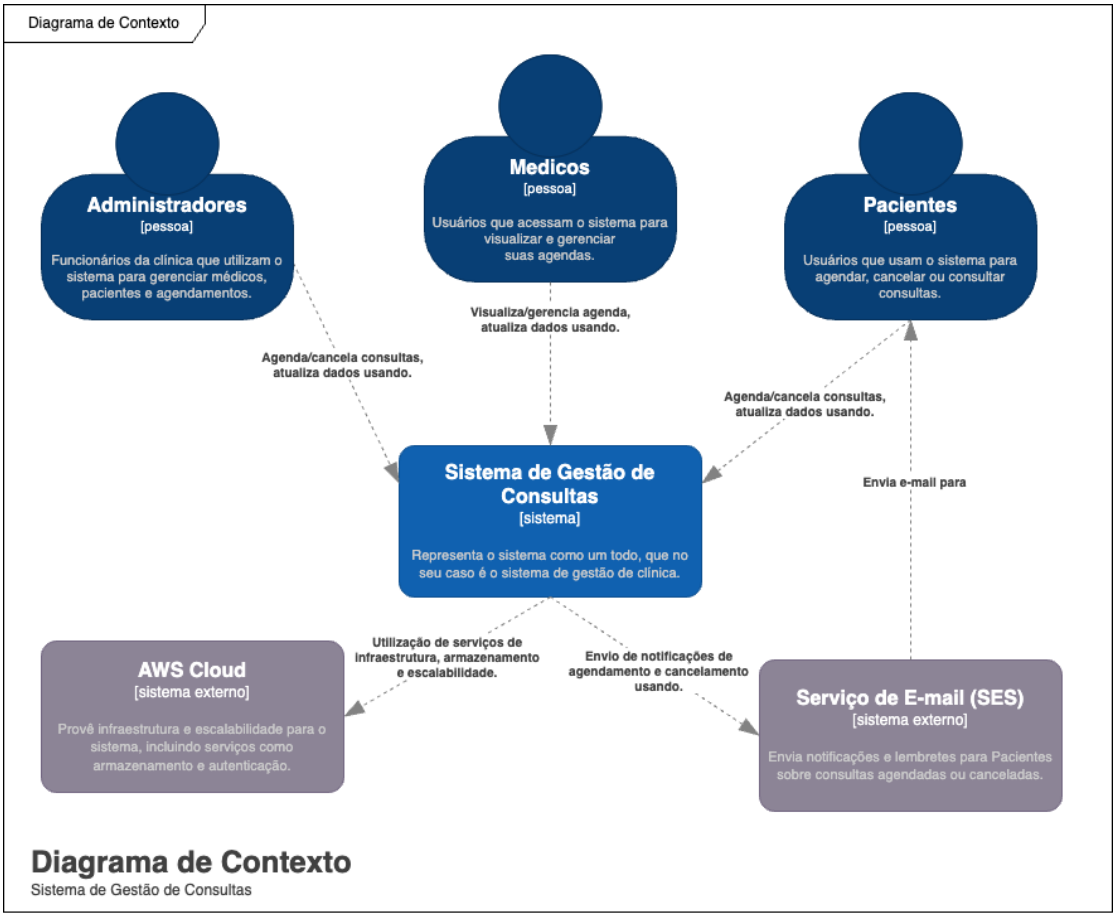

Figura 1 - Visão Geral da Solução

A Figura 2 apresenta o diagrama de contexto da solução proposta, com todos os principais módulos e interfaces, ilustrando a macroarquitetura do sistema de agendamento de consultas médicas.

A solução permite a interação de três tipos de usuários principais:

- Médicos: Podem acessar o aplicativo móvel para gerenciar seu perfil, visualizar suas consultas e receber notificações de agendamento ou cancelamento.
- Pacientes: Têm acesso ao auto-cadastramento, agendamento de consultas, cancelamento, visualização do histórico de consultas e notificações de lembrete.
- Funcionários/Admin: Podem realizar o cadastro de médicos e pacientes, agendar e cancelar consultas, gerenciar perfis e visualizar dados de consulta para suporte aos demais usuários.

Esses usuários interagem com a Aplicação Móvel, que atua como a Interface de Usuário, oferecendo uma navegação acessível e amigável. A aplicação móvel comunica-se diretamente com a API da Aplicação Central, desenvolvida em Java com o padrão MVC e implementada como uma aplicação backend monolítica.

A Aplicação Central processa todas as regras de negócio e validações, como verificações de horário de funcionamento, inatividade de médicos ou pacientes, e condições de cancelamento. Esta aplicação central está integrada a diversos módulos de serviço e componentes, incluindo:

- Base de Dados (AWS RDS PostgreSQL): Armazena todas as informações essenciais, como cadastros de médicos e pacientes, consultas agendadas, histórico de cancelamentos e logs de auditoria. A comunicação com o banco de dados é feita via JPA/Hibernate para assegurar integridade e consistência de dados.
- Serviço Externo de Notificação (AWS SNS ou SendGrid): Responsável pelo envio de notificações automáticas para médicos e pacientes, incluindo confirmações, cancelamentos e lembretes de agendamento com antecedência de um dia, proporcionando um canal de comunicação confiável e imediato.

Este diagrama oferece uma visão simplificada, porém completa, da arquitetura de uma aplicação monolítica centralizada, estruturada para facilitar a futura migração para uma arquitetura de microsserviços.

3.4 Diagrama de Casos de Uso

Figura 2 – Diagrama de Caso de Uso

Atores Principais

- Médico: Realiza ações de gerenciamento de seu próprio cadastro (parcial) e consulta seus agendamentos.
- Paciente: Realiza ações de auto-cadastramento, agendamento, cancelamento de consultas e consulta de seu histórico de consultas.

- Admin (Funcionário/Admin): Responsável pelo gerenciamento dos cadastros de médicos e pacientes e pela administração de agendamentos e cancelamentos conforme necessário.
- Sistema Externo de Notificação: Serviço externo utilizado para envio de notificações de lembrete e confirmação de agendamento/cancelamento via e-mail/SMS para médicos e pacientes.

Casos de Uso Principais

- Cadastrar Médico: Permite ao Admin cadastrar um novo médico no sistema, incluindo dados como nome, CRM, especialidade e endereço.
- Listar Médicos: O Admin pode listar os médicos cadastrados, com opções de visualização de informações detalhadas como nome, especialidade, CRM e status (ativo/inativo).
- Atualizar Médico: Permite ao Admin ou ao Médico atualizar dados específicos do cadastro do médico, com restrições sobre o e-mail, CRM e especialidade.
- Inativar Médico: Permite ao Admin inativar um médico, fazendo com que ele não possa ser selecionado para novos agendamentos, sem excluir seus dados do sistema.
- Cadastrar Paciente: Permite ao Admin ou ao próprio Paciente realizar o auto-cadastramento no sistema, incluindo informações como nome, CPF, dados de contato e endereço.
- Listar Pacientes: O Admin pode listar todos os pacientes cadastrados, visualizando informações básicas como nome, e-mail e CPF.Permitir ao Admin ou ao Paciente atualizar dados específicos do cadastro do paciente, com restrições sobre o e-mail e o CPF.
- Inativar Paciente: Permite ao Admin inativar um paciente, evitando que ele realize novos agendamentos, sem excluir seus dados do sistema.
- Agendar Consulta: Permite ao Paciente ou ao Admin agendar uma consulta com um Médico disponível, respeitando as regras de horário, disponibilidade e inatividade. Caso um médico específico não seja selecionado, o sistema realiza uma seleção aleatória de um médico disponível.
- Cancelar Consulta: Permite ao Paciente ou ao Admin cancelar uma consulta com um período mínimo de antecedência de 24 horas, solicitando um motivo para o

cancelamento (por exemplo, desistência do paciente ou cancelamento pelo médico).

- Notificar Agendamento e Cancelamento: Após o agendamento ou cancelamento de uma consulta, o sistema envia uma notificação automática para o paciente e o médico, utilizando o serviço externo para o envio de e-mails e/ou SMS. Notificações incluem confirmação de consulta, avisos de cancelamento, e lembretes de agendamento com um dia de antecedência.

3.5 Diagrama de Classe de domínio

O Diagrama de Classe de Domínio apresentado na Figura 4 ilustra a estrutura principal das classes do sistema de agendamento de consultas médicas, representando as entidades e suas relações dentro do contexto do modelo de dados da aplicação.

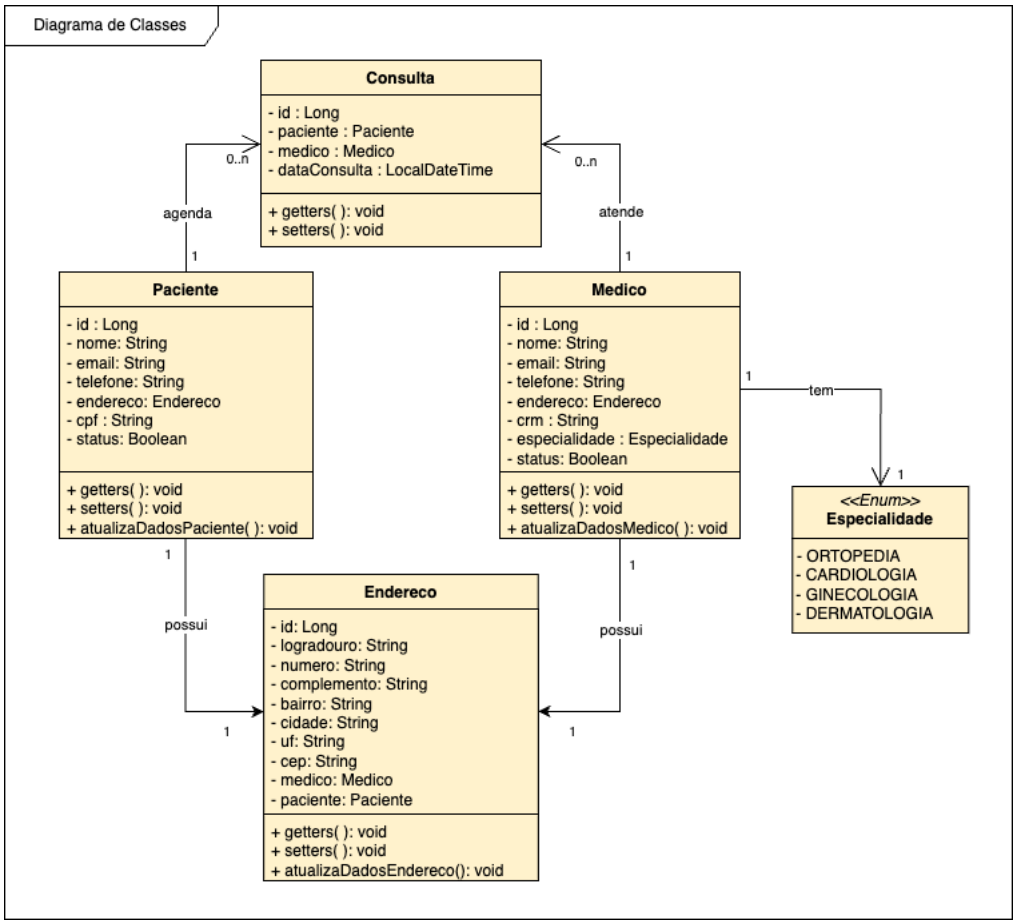

Figura 3 – Diagrama de Classe de Domínio

Este diagrama é fundamental para entender como as principais classes do sistema estão interconectadas e como os dados fluem entre elas. Ele inclui as entidades que representam os conceitos centrais do domínio, como Médico, Paciente, Consulta, Endereço, entre outras.

As classes são estruturadas de acordo com as regras de negócio e os requisitos funcionais definidos para o sistema. A seguir, uma descrição das principais classes e seus relacionamentos:

- Médico: Representa o profissional da saúde, com atributos como nome, CRM, especialidade, e endereço. A classe Médico está associada a várias Consultas que ele realiza.
- Paciente: Representa o paciente do sistema, com dados como nome, CPF, e endereço. Um paciente pode ter múltiplas Consultas agendadas.
- Consulta: Representa o agendamento de uma consulta médica, com atributos como data, hora, status (agendada, cancelada, concluída), e as entidades associadas ao agendamento, como Médico e Paciente. Esta classe também é responsável pela verificação das regras de negócio relacionadas ao horário e disponibilidade.
- Endereço: Tanto o Médico quanto o Paciente possuem um Endereço, representado por esta classe, que contém atributos como rua, número, bairro, cidade e estado.

Além dessas, o diagrama pode incluir outras classes de suporte para a modelagem do sistema, como Notificação (para o envio de mensagens), Auditoria (para registros de alterações), e Configuração (para parâmetros de agendamento, como horários e políticas de cancelamento).

O diagrama de classe de domínio serve como a base para o desenvolvimento da camada de persistência de dados do sistema e define claramente as entidades que serão manipuladas dentro da lógica do aplicativo. Ele proporciona uma visão clara das interações entre as entidades e a estrutura que suportará os processos de agendamento e gerenciamento de consultas médicas de forma eficiente e escalável.

Link aplicação Github: *https://github.com/cleber-campos/tcc-app-med-control*

3.6 Arquitetura de Dados

A arquitetura de dados desta aplicação foi projetada para garantir uma gestão eficiente, segura e escalável das informações, suportando o fluxo contínuo de dados entre os diferentes módulos do sistema de agendamento de consultas médicas. A escolha de tecnologias, como o AWS RDS PostgreSQL, facilita a integridade e a consistência dos dados, enquanto proporciona flexibilidade para uma eventual migração para uma arquitetura de microsserviços.

Modelagem de Dados Relacional

A modelagem de dados do sistema é baseada em um banco de dados relacional, utilizando o **AWS RDS PostgreSQL** para o armazenamento das informações. O modelo foi desenvolvido para refletir as principais entidades da aplicação, como **Médicos**, **Pacientes** e **Consultas**, sendo estruturado de forma a atender às necessidades de gerenciamento e agendamento de consultas médicas.

Abaixo, apresentamos o diagrama de modelagem relacional, que ilustra as principais entidades e seus relacionamentos.

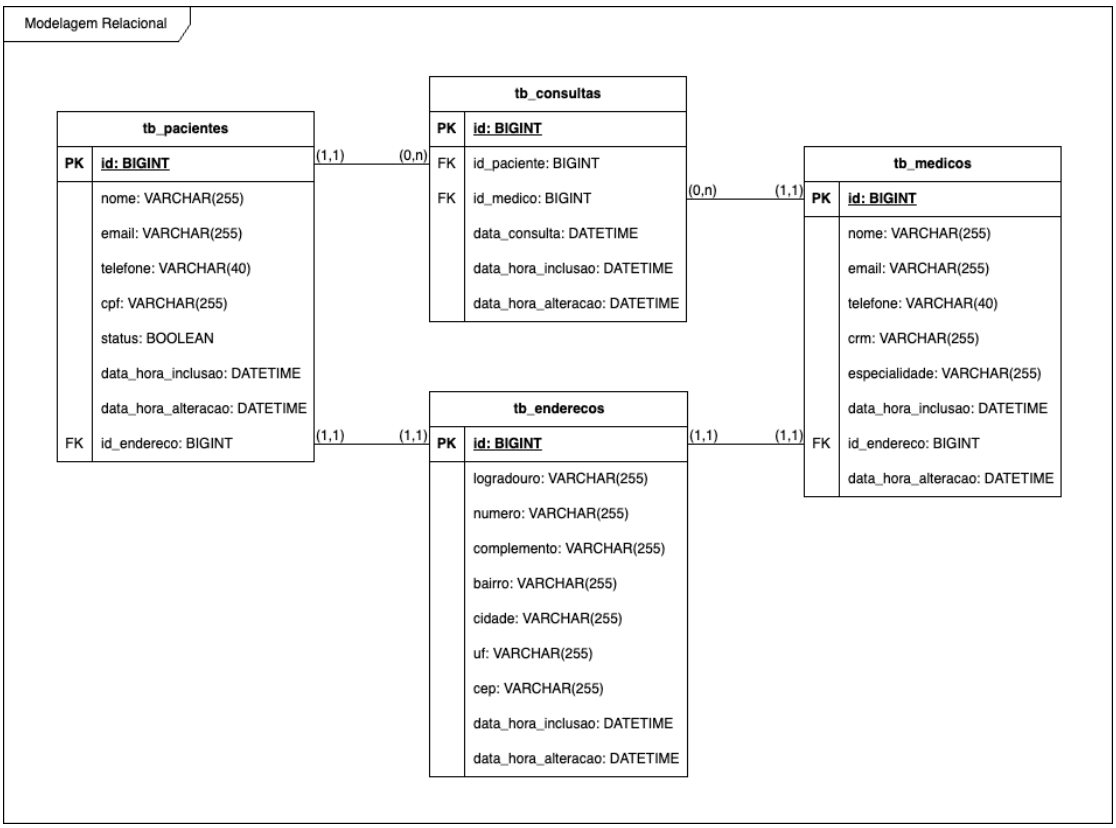

Figura 5 – Diagrama de Modelagem Relacional

Entidades Principais

- Médico: A classe Médico contém informações essenciais sobre o profissional da saúde, como:
  - id: Identificador único do médico (PK).
  - nome: Nome do médico.
  - email: Endereço de e-mail para comunicação.
  - telefone: Número de telefone para contato.
  - CRM: Número de registro profissional, único para cada médico.
  - especialidade: A especialidade médica (por exemplo, Cardiologia, Ortopedia).
  - endereco: Endereço completo do médico, que é modelado como uma entidade separada, associando dados como rua, número, bairro, cidade e estado.

- Paciente: A classe Paciente armazena os dados do paciente, que são essenciais para o agendamento de consultas e para o gerenciamento de seu histórico médico.
  - id: Identificador único do paciente (PK).
  - nome: Nome do paciente.
  - email: E-mail do paciente.
  - telefone: Número de telefone para contato.
  - cpf: CPF, um identificador único para cada paciente.
  - endereco: Endereço completo do paciente, modelado separadamente, semelhante ao do médico.

- Consulta: A classe Consulta gerencia o agendamento das consultas médicas.
  - id: Identificador único da consulta (PK).
  - id_paciente: Chave estrangeira (FK) que referencia o paciente que agendou a consulta.
  - id_medico: Chave estrangeira (FK) que referencia o médico que realizará a consulta.
  - data_hora: A data e hora do agendamento da consulta.
  - status: Status da consulta (agendada, cancelada, realizada).

Relacionamentos

- Médico e Consulta: A relação entre Médico e Consulta é de um para muitos, ou seja, um médico pode atender várias consultas, mas cada consulta está associada a um único médico.
- Paciente e Consulta: A relação entre Paciente e Consulta também é de um para muitos, onde um paciente pode ter várias consultas, mas cada consulta é vinculada a um único paciente.

Chaves Primárias e Estrangeiras

- Chaves Primárias (PK): Cada entidade, como Médico, Paciente e Consulta, possui um campo id que funciona como identificador único, garantindo que cada registro seja único dentro da tabela.
- Chaves Estrangeiras (FK): A tabela Consulta contém duas chaves estrangeiras:
- id_paciente: Referencia a tabela Paciente, associando cada consulta ao paciente correspondente.
- id_medico: Referencia a tabela Médico, vinculando cada consulta ao médico responsável pelo atendimento.

Integridade e Restrições de Dados

- Integridade Referencial: As chaves estrangeiras entre as entidades garantem a integridade referencial, ou seja, asseguram que as consultas estejam sempre associadas a médicos e pacientes válidos.
- Restrições de Unicidade: Existem restrições de unicidade para campos como CRM em Médico e CPF em Paciente, para garantir que esses dados sejam únicos e evitar duplicação.
- Regras de Negócio: Regras específicas de negócio são aplicadas no nível da aplicação para garantir o correto funcionamento do sistema, como verificação de horário de funcionamento, intervalo entre consultas e a verificação de disponibilidade de médicos e pacientes.

Persistência e Acesso a Dados

O sistema utiliza o Spring Data JPA para gerenciar a persistência dos dados e realizar o mapeamento objeto-relacional (ORM). Por meio do JPA, as entidades Java são mapeadas para as tabelas correspondentes no banco de dados, simplificando a manipulação de dados através de operações CRUD. O uso de JPA Repositories proporciona uma maneira fácil e eficiente de realizar consultas e modificações no banco de dados, além de permitir a personalização de consultas complexas quando necessário.

Segurança e Controle de Acesso

Para garantir a proteção e o controle adequado dos dados, a arquitetura de dados implementa mecanismos de segurança em várias camadas:

- Autenticação e Autorização: O serviço AWS Cognito é integrado ao sistema para gerenciar autenticação e autorização, garantindo que apenas usuários autenticados possam acessar dados sensíveis e realizar ações no sistema.
- Criptografia: Todos os dados em trânsito entre a aplicação e o banco de dados são criptografados utilizando TLS (Transport Layer Security), garantindo a proteção contra interceptação de informações sensíveis, como CPF e histórico de consultas.
- Controle de Acesso Baseado em Perfis: O acesso a dados críticos, como informações de médicos, pacientes e consultas, é controlado por perfis de usuário (administrador, médico, paciente). Esse controle assegura que somente usuários autorizados possam visualizar ou modificar dados confidenciais, minimizando o risco de vazamento ou acesso não autorizado.

Essa arquitetura de dados proporciona a base necessária para o funcionamento eficiente e seguro do sistema, permitindo que a aplicação escale e se mantenha conforme os requisitos de segurança e desempenho exigidos para um sistema de agendamento médico robusto.

3.7 Protótipo de Tela

O protótipo do aplicativo mobile foi desenvolvido com o objetivo de ilustrar as principais interfaces e interações do usuário, oferecendo uma visualização inicial da experiência de uso e das funcionalidades-chave. Ele permite avaliar a usabilidade e verificar a disposição de componentes antes da implementação. Abaixo, são descritas as telas e funcionalidades que compõem o protótipo:

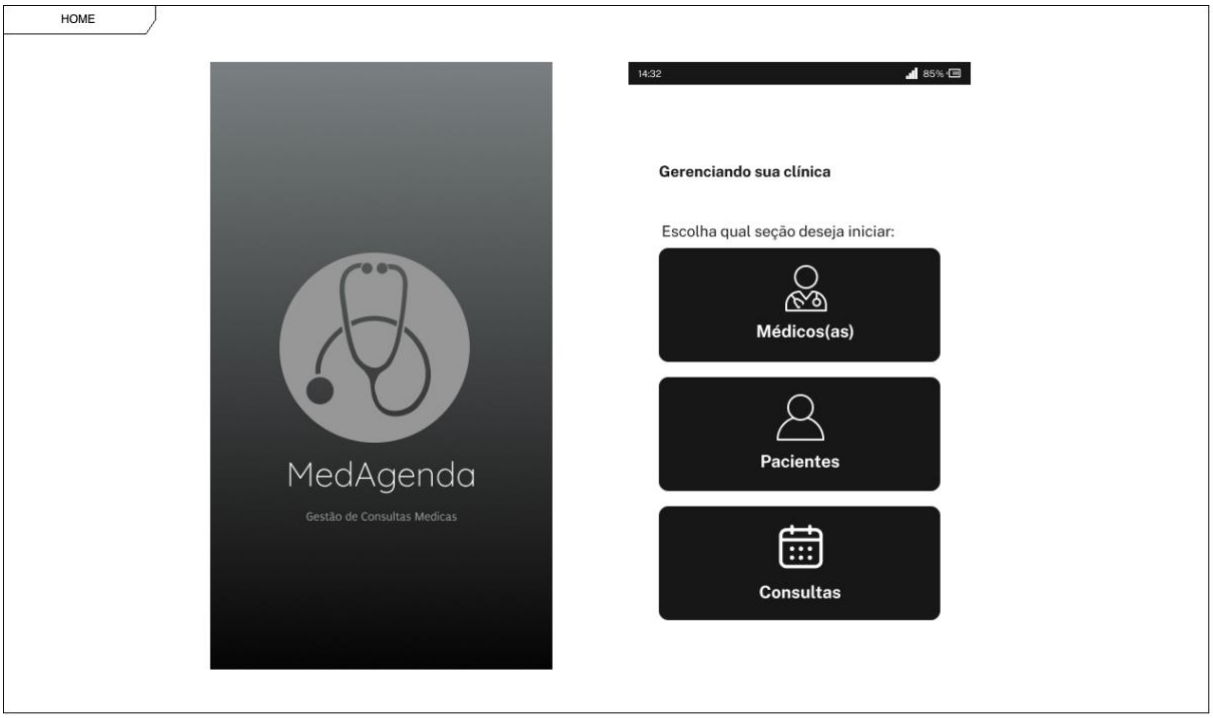

Figura 6 – Protótipo de tela – Home

A tela inicial ou Home serve como ponto de acesso principal para as funcionalidades do aplicativo. Nela, o usuário pode navegar para as telas de gerenciamento de pacientes, médicos e consultas. Essa interface também fornece um resumo das funcionalidades disponíveis no aplicativo.

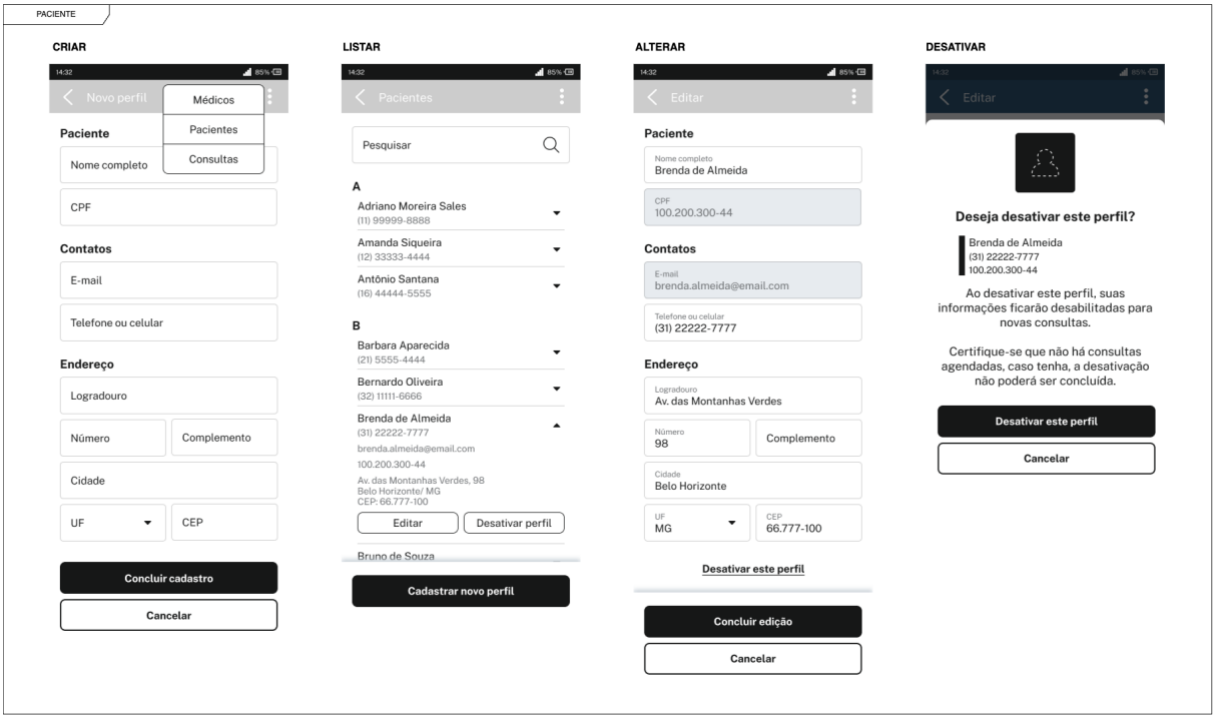

Figura 7 – Protótipo de tela – Criar, alterar, listar e desativar paciente

Nesta tela, o usuário pode realizar ações relacionadas ao cadastro de pacientes, como criar, alterar, listar e desativar registros. Os botões de ação e os campos de formulário são dispostos de forma a simplificar o fluxo de gerenciamento de pacientes.

Propósito: Facilitar o gerenciamento de informações dos pacientes.

Funcionalidades principais: Inclusão, edição, listagem e desativação de pacientes.

Elementos de interface: Campos para nome, CPF, contato e endereço, botões de confirmação e navegação.

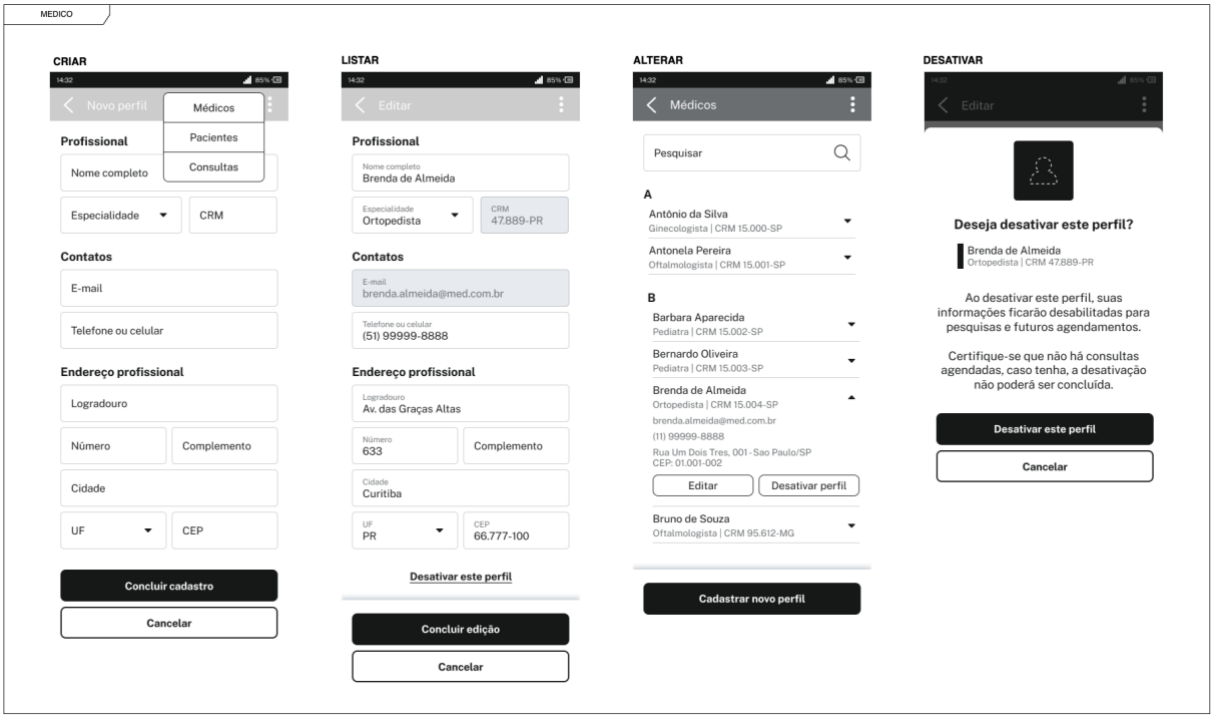

Figura 8 – Protótipo de tela – Criar, alterar, listar e desativar medico

A tela de gerenciamento de médicos permite ações como criação, alteração, listagem e desativação de registros. Esta tela possui campos específicos, como o CRM e a especialidade do médico, que são essenciais para o controle de informações da equipe médica.

Propósito: Gerenciar dados dos médicos vinculados ao sistema.

Funcionalidades principais: Cadastro, edição, listagem e desativação de médicos.

Elementos de interface: Campos de nome, CRM, especialidade e contato, botões de navegação e confirmação.

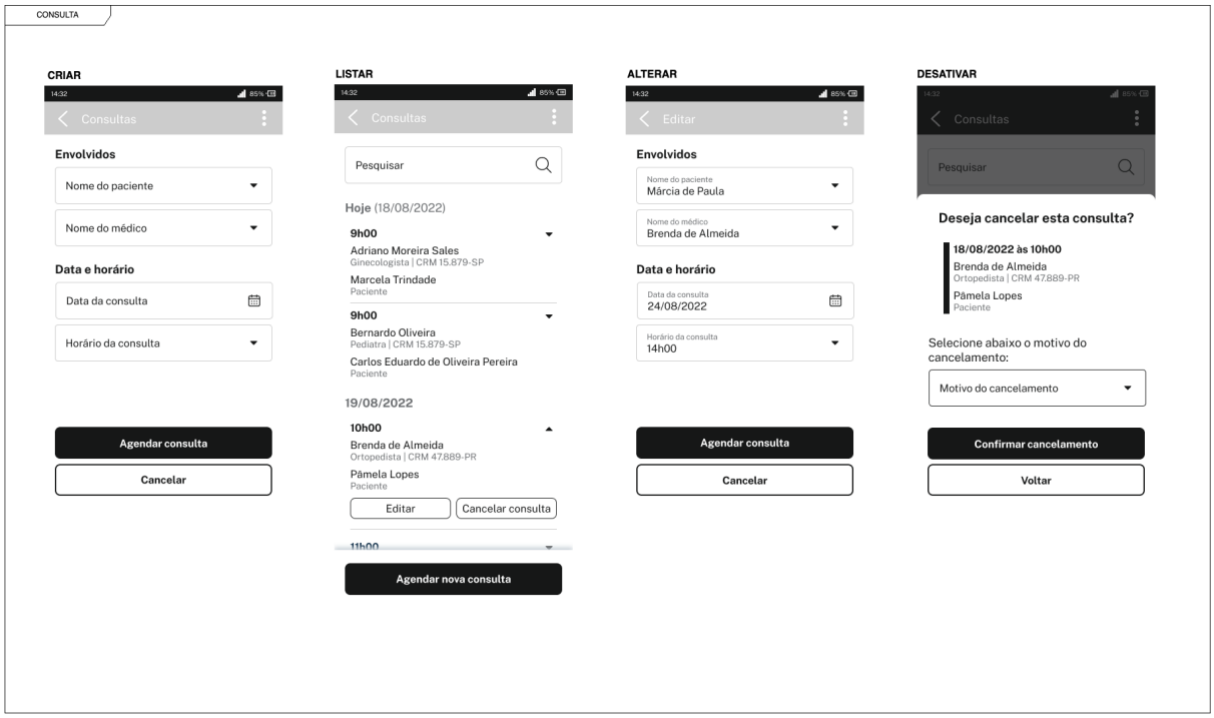

Figura 9 – Protótipo de tela – Criar, alterar, listar e cancelar consulta

Na tela de agendamento, o usuário pode criar, listar e cancelar consultas médicas. Esta interface permite a seleção de médicos e pacientes, e a escolha de data e hora para a consulta, respeitando a disponibilidade dos profissionais.

Propósito: Realizar e gerenciar agendamentos de consultas.

Funcionalidades principais: Inclusão, edição, listagem e cancelamento de consultas.

Elementos de interface: Seleção de médico e paciente, calendário para escolher data, hora, e botão de confirmação.

Funcionalidades de Navegação e Interatividade

A navegação entre as telas foi projetada para ser fluida e intuitiva. Ícones e botões de fácil compreensão foram usados para facilitar o acesso a diferentes funcionalidades. Elementos interativos, como os botões de confirmação e os componentes de data e seleção, estão organizados de forma acessível para aprimorar a experiência do usuário.

Link do protótipo: *https://www.figma.com/proto/lmv8xwR1dAan8vc7ztpyLW/app-agend-consulta?node-id=2-1008&node-type=canvas&t=GCiXgnISvX8WUH0U-1&scaling=scale-down&content-scaling=fixed&page-id=2%3A1007&starting-point-node-id=2%3A1008&share=1*

4. Modelagem Arquitetural

Esta seção apresenta a modelagem arquitetural da solução proposta para o sistema de agendamento de consultas médicas. O objetivo é fornecer uma visão clara e completa da estrutura do sistema, garantindo que todas as decisões arquiteturais sejam bem fundamentadas e que o sistema seja facilmente escalável, desde a fase de prova de conceito até a sua implementação final.

Para documentar a arquitetura do sistema, foi escolhido o modelo C4, uma abordagem amplamente utilizada para descrever e visualizar sistemas de software. O modelo C4 se baseia na decomposição do sistema em diferentes níveis de abstração, oferecendo uma representação clara e concisa da arquitetura, por meio de diagramas detalhados e descrições.

O modelo C4 é composto por quatro níveis hierárquicos de abstração:

- **Nível 1: Diagrama de Contexto**

Este diagrama oferece uma visão geral do sistema, destacando suas interações com os usuários e outros sistemas externos, como os serviços de autenticação via AWS Cognito e serviços de envio de e-mails (por exemplo, Amazon SES ou SendGrid). Ele mostra como o sistema se conecta com os usuários (administradores, médicos, pacientes) e com as infraestruturas externas, incluindo a base de dados no AWS RDS PostgreSQL.

- **Nível 2: Diagrama de Containers**

No diagrama de containers, são apresentados os principais containers que compõem o sistema, como os aplicativos que gerenciam o front-end e back-end, os containers responsáveis pela persistência de dados (utilizando JPA e Spring Data), e os serviços externos integrados (autenticação e envio de notificações). O diagrama detalha como esses containers se interagem e compartilham dados, assegurando a comunicação e integração entre o cadastro-service e o agendamento-service, componentes chave para a funcionalidade do sistema.

- **Nível 3: Diagrama de Componentes**

Este diagrama detalha os componentes internos de cada container, explicando como eles colaboram para realizar as funcionalidades principais da aplicação, como o cadastro de médicos e pacientes, o agendamento de consultas, a validação das regras de negócio e o envio de notificações. No cadastro-service, são descritos os componentes responsáveis pelo CRUD de médicos e pacientes, enquanto no agendamento-service, o foco é na lógica de agendamento e cancelamento de consultas, incluindo as regras de disponibilidade de médicos e horários de funcionamento.

- **Nível 4: Diagrama de Código**

Este diagrama, que será abordado na próxima seção (Seção 5), detalha a estrutura de classes, funções e outros elementos de código necessários para implementar a lógica da aplicação, como o controle de acesso, validação de dados e integração com a base de dados. A descrição de como os componentes internos são implementados será discutida em detalhes na seção seguinte.

A adoção do modelo C4 para a documentação da arquitetura do sistema facilita a visualização e compreensão do fluxo de informações e da estrutura de interação entre os diferentes módulos. Ao fornecer uma visão clara desde a interação com o usuário até a implementação dos componentes internos, a modelagem ajuda a promover uma comunicação eficaz entre as equipes de desenvolvimento. Além disso, essa abordagem é crucial para a evolução do sistema, especialmente considerando a migração futura para uma arquitetura de microsserviços.

Essa modelagem arquitetural visa não apenas apoiar o desenvolvimento inicial, mas também garantir que o sistema seja flexível o suficiente para futuras expansões e melhorias, como a inclusão de novos módulos ou a adoção de microsserviços.

4.1 Diagrama de Container

O Diagrama de Containers oferece uma visão mais detalhada da arquitetura do sistema, mostrando como os diferentes componentes (containers) da aplicação são distribuídos e interagem entre si. A Figura 10 ilustra a estrutura do sistema em termos das partes principais, como a aplicação móvel, o backend, o banco de dados, serviços externos, e como eles se comunicam.

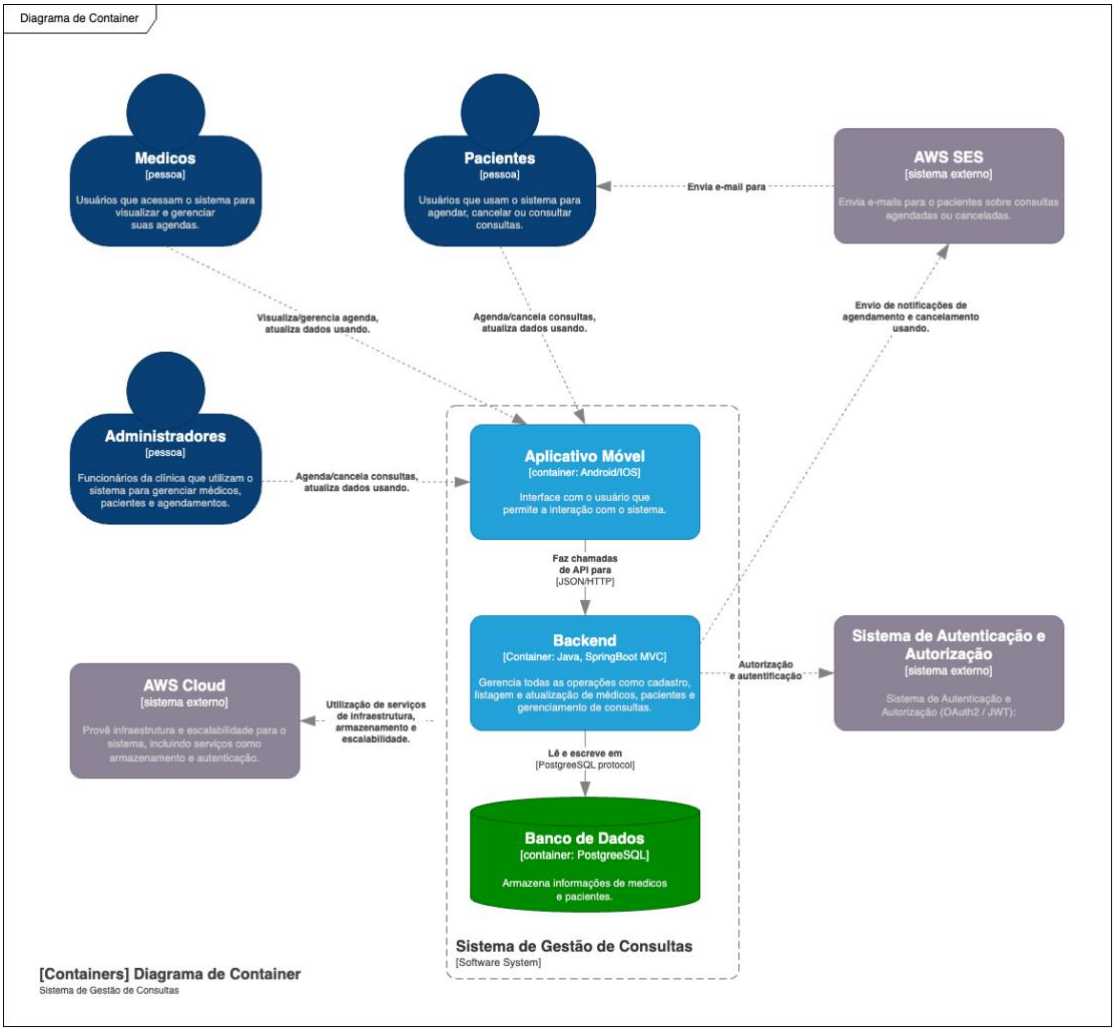

Figura 10 – Diagrama de container

Para a aplicação de agendamento de consultas médicas, a arquitetura será inicialmente monolítica, com a possibilidade de migração futura para uma arquitetura de microsserviços, conforme o crescimento e a necessidade de escalabilidade do sistema. Abaixo estão descritos os containers principais que compõem a solução:

- Aplicativo Móvel (iOS/Android): Este container é responsável pela interface de usuário, atendendo médicos, pacientes e funcionários da clínica. O aplicativo proporciona funcionalidades como cadastro de médicos e pacientes, agendamento e cancelamento de consultas, visualização de informações e envio de notificações. A comunicação entre o aplicativo e o backend será feita via API RESTful, utilizando uma camada de comunicação segura (HTTPS).

  Tecnologia: Aplicativo nativo para iOS e Android (React Native ou Kotlin/Swift).

- Backend (Java Spring Boot): O backend é o componente central do sistema, responsável por processar as solicitações vindas do aplicativo móvel e da interface de usuário. Ele gerencia o cadastro, listagem e atualização de médicos e pacientes, além do agendamento, cancelamento de consultas e validação de regras de negócio. O backend será desenvolvido utilizando Java com o framework Spring Boot, estruturado no padrão MVC, e fará a comunicação com o banco de dados relacional utilizando JPA (Hibernate).

  Tecnologia: Java, Spring Boot, Spring MVC, JPA (Hibernate).

- Banco de Dados Relacional (PostgreSQL): O sistema usará um banco de dados relacional para armazenar informações sobre médicos, pacientes, consultas e dados relacionados. O banco de dados será hospedado na AWS através do serviço RDS, garantindo alta disponibilidade, segurança e escalabilidade.

  Tecnologia: PostgreSQL, hospedado na AWS RDS.

- Serviço de Notificação (Amazon SES): O serviço de notificações será utilizado para enviar alertas aos pacientes e médicos sobre o agendamento de consultas. As notificações podem ser enviadas por meio de push notifications no aplicativo ou por e-mail. O sistema integrará com o Amazon SES para o envio de e-mails, configurado na AWS.

  Tecnologia: Amazon SES.

- Sistema de Autenticação e Autorização (OAuth2 / JWT): Para garantir a segurança da aplicação, será implementado um sistema de autenticação e autorização baseado em tokens JWT (JSON Web Tokens). Isso permitirá que médicos, pacientes e administradores se autentiquem de forma segura. O backend gerenciará o processo de login, geração de tokens e validação de permissões.

Tecnologia: OAuth2, JWT (Spring Security).

- Plataforma de Nuvem (AWS): A infraestrutura do sistema será hospedada na AWS, utilizando uma série de serviços para garantir escalabilidade, segurança e alta disponibilidade. Além do banco de dados RDS, serão utilizados serviços como AWS Lambda (para funções serverless, se necessário), S3 (para armazenamento de arquivos), e EC2 (para hospedar o backend).

  Tecnologia: AWS (EC2, RDS, Lambda, S3).

No diagrama de containers, a comunicação entre esses componentes será detalhada, destacando como o aplicativo móvel interage com o backend via API RESTful, como o backend acessa o banco de dados para persistir e consultar dados, e como os serviços de notificação são acionados para enviar lembretes e confirmações.

4.2 Diagrama de Componentes

O Diagrama de Componentes, conforme ilustrado na Figura 11, apresenta os principais componentes envolvidos na solução do sistema de agendamento de consultas médicas. A seguir, descrevemos cada um desses componentes e suas responsabilidades:

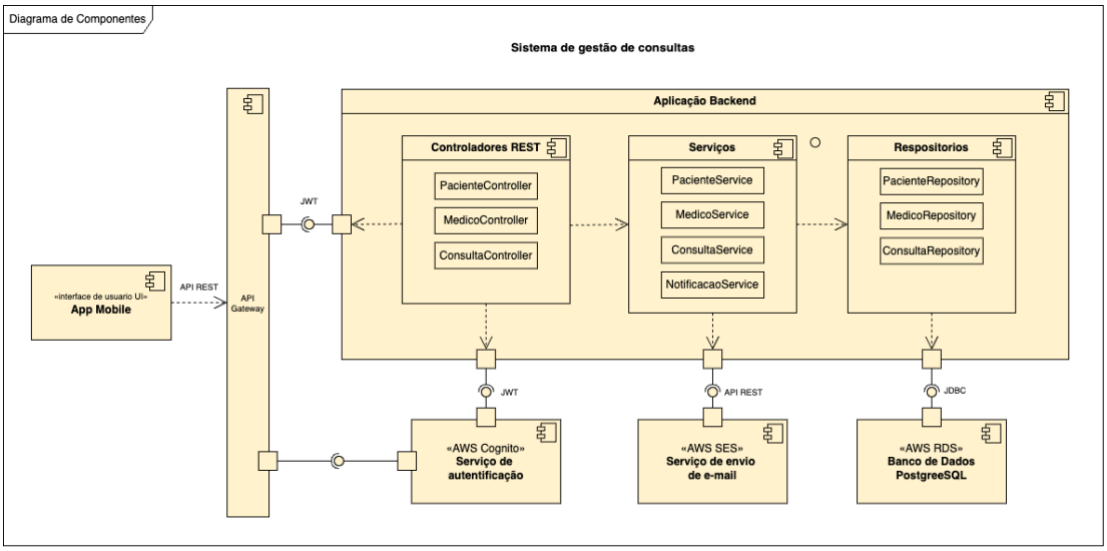

Figura 11 – Diagrama de Componentes

App Mobile (UI): O Aplicativo Móvel serve como a interface de usuário, acessível por dispositivos móveis (Android/iOS). Ele permite que médicos, pacientes e funcionários da clínica interajam com o sistema de forma intuitiva e eficiente. O aplicativo oferece funcionalidades para o cadastro de médicos e pacientes, agendamento de consultas, consulta de dados e notificações. A comunicação com o backend será realizada via APIs RESTful.

Tecnologia: React Native ou Kotlin/Swift.

Controladores REST: Os controladores são responsáveis por gerenciar as requisições de entrada e saída, realizando a mediação entre o frontend e o backend.

- MedicoController: Gerencia as operações de cadastro, listagem, atualização e inativação dos registros de médicos no sistema.
- PacienteController: Gerencia as operações de cadastro, listagem, atualização e inativação dos pacientes no sistema.
- ConsultaController: Gerencia o agendamento e cancelamento de consultas, aplicando as regras de negócio como validação de horários, disponibilidade de médicos, e antecedência mínima para o agendamento.

Serviços: Os serviços implementam a lógica de negócios, orquestrando as operações de manipulação de dados e regras de validação.

- MedicoService: Contém a lógica de negócios para o gerenciamento de médicos, incluindo validações de entrada e manipulação dos dados de médicos.
- PacienteService: Contém a lógica de negócios para o gerenciamento de pacientes, incluindo validações de entrada e manipulação dos dados de pacientes.
- ConsultaService: Gerencia a lógica para agendamento e cancelamento de consultas, garantindo que todas as regras relacionadas a horários, disponibilidade e restrições sejam cumpridas.
- NotificacaoService: Responsável por enviar notificações sobre consultas para médicos e pacientes, integrando com o serviço de notificação externo, como o Amazon SNS ou Amazon SES.

Repositórios: Os repositórios são responsáveis pelas operações de persistência de dados no banco de dados.

- MedicoRepository: Interface que define as operações de persistência para os dados dos médicos, utilizando o banco de dados PostgreSQL.
- PacienteRepository: Interface que gerencia as operações de persistência dos dados dos pacientes no banco de dados PostgreSQL.
- ConsultaRepository: Interface que gerencia as operações de persistência relacionadas às consultas agendadas no banco de dados PostgreSQL.

Banco de Dados (AWS RDS - PostgreSQL): O banco de dados PostgreSQL, hospedado no AWS RDS, é responsável por armazenar os dados persistentes do sistema, como médicos, pacientes e consultas. Ele garante alta disponibilidade, segurança e escalabilidade para a aplicação.

Serviço Externo de Notificação (AWS SNS/SES): O Serviço de Notificação envia atualizações automáticas sobre o status das consultas (como agendamentos e cancelamentos) para médicos e pacientes. Esse serviço se integra ao sistema via APIs, permitindo o envio de notificações por e-mail (via SES) ou SMS (via SNS).

Fluxo de Comunicação:

- App Mobile interage com o backend via API RESTful, enviando e recebendo dados através dos Controladores REST.
- Os Controladores REST invocam os Serviços correspondentes (como MedicoService, PacienteService, ConsultaService) para aplicar a lógica de negócios.
- Os Serviços interagem com os Repositórios para persistir ou consultar dados no Banco de Dados.
- O NotificacaoService integra com o serviço externo (AWS SES/SNS) para enviar notificações sobre consultas.

5. *Arquitetura de implantação em nuvem*

A arquitetura de implantação em nuvem da aplicação utiliza serviços gerenciados da AWS para garantir escalabilidade, segurança e alta disponibilidade. Abaixo, apresentamos uma visão geral dos componentes da AWS necessários para o funcionamento do sistema de agendamento de consultas médicas.

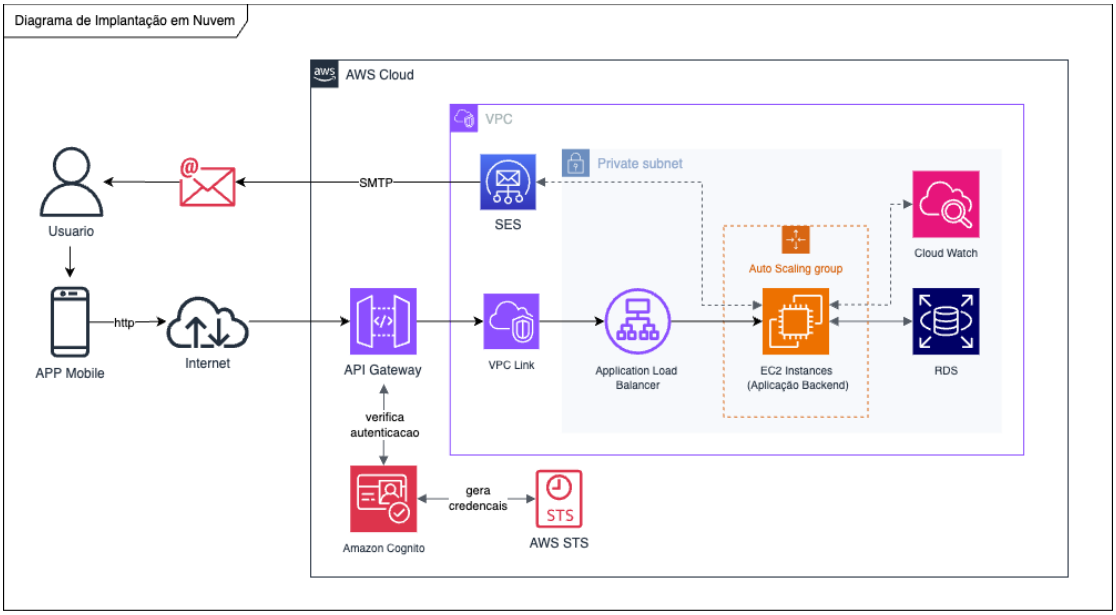

Figura 12 – Diagrama de Implantação em Nuvem

Amazon VPC (Virtual Private Cloud)

- VPC isola toda a infraestrutura da aplicação, garantindo segurança e controle sobre os recursos.
- A configuração de sub-redes públicas e privadas é feita para segregar o tráfego interno e externo:
- Sub-redes Públicas: Contêm recursos acessíveis externamente, como o API Gateway e o Load Balancer.
- Sub-redes Privadas: Contêm os serviços de back-end, como as instâncias EC2 e o banco de dados RDS, que são protegidos de acessos externos diretos, aumentando a segurança do sistema.

VPC Link

- Estabelece uma conexão segura e direta entre o API Gateway e o Application Load Balancer (ALB). Essa configuração facilita o roteamento de tráfego de forma segura entre os serviços da aplicação, sem expor diretamente a rede interna.

Application Load Balancer (ALB)

- O ALB distribui automaticamente as requisições dos usuários entre as instâncias EC2 que hospedam a aplicação.
- Ele gerencia o tráfego da camada de aplicação, equilibrando a carga entre as instâncias e garantindo alta disponibilidade e escalabilidade da aplicação.

Instâncias EC2 (Elastic Compute Cloud)

- As instâncias EC2 hospedam a aplicação Java com Spring MVC, responsável pelas operações de CRUD e agendamento de consultas.
- Elas estão configuradas em um Auto Scaling Group para escalabilidade automática, garantindo que a aplicação consiga atender a picos de tráfego sem perder desempenho.
- As instâncias EC2 se conectam a outros serviços da AWS, como AWS RDS (para armazenamento de dados), AWS Cognito (para autenticação) e AWS SES (para envio de e-mails), permitindo a integração e o bom funcionamento do sistema.

AWS RDS (Relational Database Service) - PostgreSQL

- O AWS RDS utiliza o banco de dados PostgreSQL para armazenar dados da aplicação de forma relacional.
- Configurado em uma sub-rede privada, o RDS garante a segurança dos dados armazenados e está configurado para alta disponibilidade com replicação entre regiões.
- O RDS gerencia todas as operações de persistência, como inserções, atualizações e consultas complexas, mantendo a integridade e a consistência dos dados de médicos, pacientes e consultas.

AWS Cognito

- O AWS Cognito é um serviço de autenticação e autorização que gerencia o controle de acesso para médicos, pacientes e administradores.

- Ele se integra diretamente à aplicação para fornecer tokens de autenticação, garantindo que os usuários possam acessar os endpoints protegidos de maneira segura.
- A aplicação utiliza o Cognito para validar a identidade dos usuários antes de permitir o acesso a funcionalidades sensíveis, como o agendamento de consultas.

API Gateway

- O API Gateway atua como camada intermediária entre a aplicação e o mundo externo, expondo endpoints RESTful que permitem a comunicação com a aplicação.
- Ele gerencia e filtra as requisições externas, direcionando-as para serviços como o AWS SES (para e-mails), AWS Cognito (para autenticação), e os endpoints da aplicação.
- Conectado ao VPC Link, o API Gateway permite acesso seguro aos ALB e, consequentemente, às instâncias EC2, garantindo a integridade e a segurança do tráfego de dados.

AWS SES (Simple Email Service)

- O AWS SES é utilizado para o envio de e-mails relacionados a notificações de agendamentos e cancelamentos de consultas.
- O serviço envia e-mails em tempo real para pacientes e médicos sobre mudanças no status das consultas, garantindo que as partes envolvidas sejam sempre informadas.
- A comunicação entre as instâncias EC2 e o AWS SES ocorre via conexões HTTP seguras, com o API Gateway atuando como intermediário para garantir conformidade com as melhores práticas de segurança.

AWS CloudWatch

- O AWS CloudWatch é o serviço de monitoramento integrado que coleta logs e métricas das instâncias EC2, API Gateway e AWS SES.
- Ele está configurado para gerar alertas em caso de falhas ou problemas de desempenho, permitindo a análise rápida e a resposta proativa a qualquer incidente, como lentidão nas respostas ou falhas de comunicação entre os componentes da aplicação.

6. Avaliação da Arquitetura (ATAM)

Neste capítulo, é apresentada a avaliação da arquitetura desenvolvida para o projeto, utilizando o método ATAM (Architecture Tradeoff Analysis Method). O objetivo desta análise é verificar se a arquitetura projetada atende adequadamente aos requisitos e objetivos definidos pelo cliente.

A abordagem ATAM oferece uma metodologia estruturada para avaliar trade-offs na arquitetura, levando em consideração atributos de qualidade como desempenho, segurança, manutenibilidade e escalabilidade. Esse método ajuda a identificar potenciais riscos benefícios, permitindo uma compreensão mais ampla de como a arquitetura lida com diferentes cenários e requisitos críticos.

Este capítulo descreve as etapas do processo de avaliação, que incluem a identificação dos atributos de qualidade prioritários, a análise de trade-offs, e a documentação dos resultados, garantindo que a arquitetura esteja alinhada com as expectativas do cliente e preparada para suportar futuras evoluções do sistema.

6.1. Análise das abordagens arquiteturais

Para analisar a proposta arquitetural do projeto de agendamento de consultas médicas, utilizamos o método ATAM (Architecture Tradeoff Analysis Method), que permite avaliar a solução em relação aos principais atributos de qualidade esperados. A análise é feita por meio de cenários representativos, que ilustram como a arquitetura lida com atributos como escalabilidade, segurança, disponibilidade, desempenho, manutenibilidade, usabilidade e monitoramento.

Atributos de Quailidade	Cenários	Importância	Complexidade
Escalabilidade	Cenário 1: O sistema deve suportar um aumento de usuários e requisições durante picos, utilizando Auto Scaling e ALB.	A	M
Segurança	Cenário 2: O sistema deve proteger dados sensíveis de usuários com	A	M

	autenticação via AWS Cognito e isolamento de redes.		
Disponibilidade	Cenário 3: O sistema deve garantir alta disponibilidade com redundância de dados no RDS e múltiplas zonas de disponibilidade.	A	M
Desempenho	Cenário 4: O sistema deve responder em até X segundos durante operações de CRUD em alta carga.	M	B
Manutenibilidade	Cenário 5: O sistema deve permitir modificações nos módulos com impacto mínimo em outros serviços, visando a migração futura para microsserviços.	M	M
Usabilidade	Cenário 6: A aplicação deve ser intuitiva e eficiente para pacientes e médicos, com foco em facilidade de uso no aplicativo.	M	B
Monitoramento	Cenário 7: O sistema deve permitir monitoramento de erros e desempenho com alertas via CloudWatch.	A	M

Tabela 4 – Atributos de Qualidade e Cenários

Contextualização dos Cenários

- **Cenário 1 (Escalabilidade)**: Avalia a capacidade da arquitetura em crescer horizontalmente, com a adição de novas instâncias **EC2** automaticamente conforme o volume de tráfego aumenta. O uso de **ALB** permite balanceamento de carga eficiente, distribuindo as requisições entre as instâncias de forma equilibrada.
- **Cenário 2 (Segurança)**: Analisa a implementação do **AWS Cognito** para garantir uma autenticação robusta e autorização segura para médicos, pacientes e administradores. Além disso, o uso de **VPC** para isolar os recursos críticos da aplicação, como o **RDS** e as instâncias **EC2**, garante a segurança do tráfego e a proteção de dados sensíveis.

- **Cenário 3 (Disponibilidade)**: Avalia a configuração do **AWS RDS** para garantir alta disponibilidade dos dados, utilizando replicação e múltiplas zonas de disponibilidade. Essa abordagem assegura que o sistema continue funcionando sem interrupções mesmo em casos de falha em uma das zonas de disponibilidade.
- **Cenário 4 (Desempenho)**: Avalia o desempenho da aplicação, especialmente em operações de **CRUD**, durante picos de tráfego. A meta é garantir que as requisições sejam respondidas rapidamente, com tempos de resposta eficientes para as transações de pacientes e médicos, mesmo com alta carga de usuários simultâneos.
- **Cenário 5 (Manutenibilidade)**: Verifica a modularização da aplicação, com o objetivo de facilitar a implementação de melhorias e atualizações. Além disso, a arquitetura foi projetada com a possibilidade de migração futura para uma arquitetura de **microsserviços**, permitindo que novos componentes sejam adicionados ou modificados sem impactar diretamente outros módulos da aplicação.
- **Cenário 6 (Usabilidade)**: Focado na experiência do usuário, especialmente para o público-alvo de médicos e pacientes. A aplicação deve ser intuitiva e eficiente, com foco na simplicidade da interface, especialmente em dispositivos móveis, para facilitar a navegação e o agendamento de consultas.
- **Cenário 7 (Monitoramento)**: Avalia a capacidade do sistema de monitorar o desempenho e a integridade da infraestrutura através do **AWS CloudWatch**. Esse cenário assegura que eventuais erros sejam rapidamente identificados, e que alertas sejam gerados proativamente para que a equipe técnica tome ações corretivas.

Conclusão

Esse conjunto de cenários fornece uma visão abrangente de como a arquitetura do sistema atende aos requisitos do projeto, garantindo que os principais atributos de qualidade, como escalabilidade, segurança e desempenho, sejam atendidos de forma eficiente. A análise ATAM não apenas valida a arquitetura atual, mas também permite a identificação de possíveis pontos de melhoria para garantir a evolução do sistema nas próximas fases do projeto.

6.2. Resultados Obtidos

Esta seção apresenta uma análise dos resultados da arquitetura proposta, destacando suas forças e limitações em relação aos requisitos não funcionais estabelecidos. Os testes realizados avaliam a solução em ambientes de Teste e Homologação para garantir a conformidade com os objetivos de escalabilidade, segurança, disponibilidade, entre outros atributos essenciais.

A tabela a seguir resume o desempenho da arquitetura para cada requisito não funcional (RNF), destacando se o requisito foi atendido ou se precisa de melhorias.

Requisitos Não Funcionais	Teste	Homologação
RNF01: Disponibilidade 24/7/365.	OK	OK
RNF02: Autenticação segura com AWS Cognito.	OK	OK
RNF03: Escalabilidade automática com Auto Scaling.	OK	OK
RNF04: Alta disponibilidade do banco de dados AWS RDS com replicação e failover.	OK	OK
RNF05: Tempo de resposta máximo de 2 segundos nas operações CRUD.	OK	N.A.
RNF06: Monitoramento contínuo via CloudWatch para falhas e alertas em tempo real.	OK	OK
RNF07: Segurança de dados com isolamento em sub-redes privadas na VPC.	OK	OK
RNF08: Manutenibilidade com modularização visando futura migração para microsserviços.	OK	OK
RNF09: Integração com AWS SES para notificações de agendamento e cancelamento.	OK	N.A.

Tabela 5 – Requisitos Não Fucionais

Análise dos Resultados

Pontos Fortes:

- Disponibilidade e Escalabilidade: A arquitetura atendeu com sucesso aos principais requisitos de disponibilidade e escalabilidade, utilizando configurações robustas de Auto Scaling e AWS RDS com replicação e failover, garantindo a continuidade dos serviços e a adaptação à variação no número de usuários e requisições.
- Segurança: A segurança foi bem implementada com o uso do AWS Cognito para autenticação segura de usuários e com o isolamento de dados sensíveis, através

do uso de sub-redes privadas na VPC. Esses mecanismos garantem a proteção e integridade das informações dos pacientes e médicos.

- Modularização e Manutenibilidade: A arquitetura foi organizada por domínios, facilitando a manutenção e possibilitando uma futura migração para microsserviços, caso seja necessário escalar ainda mais o sistema. A modularidade garante que alterações em um módulo tenham impacto mínimo em outros, o que é crucial para garantir a evolução contínua da aplicação.
- Monitoramento: O uso do AWS CloudWatch para monitoramento contínuo de erros e desempenho foi eficaz. Alertas em tempo real permitem que a equipe técnica responda rapidamente a qualquer falha ou irregularidade, assegurando que a operação do sistema seja mantida sem interrupções significativas.

Limitações:

- Desempenho: Embora o sistema tenha atendido ao requisito de tempo de resposta nas operações CRUD em ambiente de teste, pode ser necessário realizar ajustes adicionais no desempenho, especialmente considerando cenários de alta carga ou picos de tráfego, para garantir que o tempo de resposta não ultrapasse o limite de 2 segundos.
- Integração com serviços externos: A integração com o AWS SES para envio de notificações de agendamento e cancelamento foi implementada, mas pode exigir mais testes em ambiente de homologação para validar a confiabilidade e desempenho desta integração sob condições de produção.

Em geral, a arquitetura do sistema de agendamento de consultas médicas demonstrou ser robusta, segura e escalável, atendendo aos requisitos não funcionais mais críticos e proporcionando uma base sólida para futuras melhorias e evolução do sistema.

7. Avaliação Crítica dos Resultados

Nesta seção, apresentamos uma avaliação crítica da arquitetura proposta para o sistema de agendamento de consultas médicas, destacando os principais pontos positivos e negativos, bem como os prós e contras das tecnologias selecionadas. Esta análise proporciona uma visão abrangente dos aspectos arquiteturais que agregam valor e dos pontos que podem impactar a escalabilidade, manutenção ou custo da solução.

Ponto avaliado	Descrição
Disponibilidade e Escalabilidade	A arquitetura utiliza o **Auto Scaling** das instâncias EC2 e a alta disponibilidade do **AWS RDS** para garantir operação contínua e escalabilidade conforme necessário, especialmente durante picos de demanda no sistema de agendamento.
Segurança	A integração com **AWS Cognito** para autenticação garante controle robusto de acesso aos dados sensíveis de médicos e pacientes. O uso da **VPC** com sub-redes privadas também oferece isolamento de rede, aumentando a segurança dos componentes internos do sistema.
Modularidade e Manutenibilidade	A organização em módulos por domínio facilita a manutenção e possibilita uma migração futura para uma arquitetura de **microsserviços**, garantindo flexibilidade para expansões e modificações com impacto mínimo.
Monitoramento e Gerenciamento	A utilização do **CloudWatch** permite o rastreamento e monitoramento contínuo das atividades da aplicação, com alertas sobre falhas, o que auxilia na manutenção preventiva e resposta rápida a incidentes.
Custo	Embora os serviços gerenciados da AWS proporcionem alta disponibilidade e segurança, o custo de operação pode aumentar substancialmente à medida que a aplicação escala, especialmente com o uso de recursos como **EC2**, **RDS** e **CloudWatch**.
Complexidade de Configuração	A configuração de segurança e o isolamento dos dados entre as sub-redes públicas e privadas adicionam uma complexidade significativa ao setup inicial da **VPC** e da infraestrutura de nuvem. Isso exige um nível de especialização em infraestrutura.

Tabela 6 – Quadro Resumo

Análise Crítica

Pontos Positivos:

- Escalabilidade e Disponibilidade: A arquitetura foi projetada para lidar com picos de demanda e garantir alta disponibilidade. O uso de Auto Scaling para instâncias EC2 e replicação e failover no AWS RDS garante que o sistema de agendamento de consultas possa escalar de forma eficiente conforme o número de usuários cresce, mantendo a continuidade dos serviços em diferentes zonas de disponibilidade.
- Segurança: A utilização do AWS Cognito para autenticação de usuários médicos e pacientes oferece um controle de acesso robusto. A separação de componentes em sub-redes privadas dentro da VPC fortalece a segurança, limitando a exposição dos dados e recursos críticos da aplicação.
- Modularidade e Evolução: A arquitetura modular organizada por domínios é uma base sólida para o crescimento do sistema. Ela facilita a migração para microsserviços, permitindo uma evolução gradual do sistema sem grandes reestruturações. Esta modularidade também facilita a manutenção contínua da aplicação à medida que novos recursos são adicionados.
- Monitoramento Proativo: O AWS CloudWatch permite monitoramento contínuo de todos os componentes do sistema, proporcionando visibilidade sobre o desempenho e permitindo que incidentes sejam detectados e resolvidos rapidamente, melhorando a confiabilidade da aplicação.

Pontos Negativos:

- Custos Crescentes: Embora os serviços gerenciados da AWS tragam benefícios como alta disponibilidade e segurança, eles também geram custos operacionais que aumentam conforme o uso e a escala da aplicação. O custo com EC2, RDS, e CloudWatch pode ser considerável, especialmente em um ambiente com alto volume de requisições de agendamento e grandes volumes de dados. Isso pode afetar a sustentabilidade econômica a longo prazo, exigindo um controle rigoroso do orçamento.
- Complexidade na Configuração Inicial: A configuração do sistema requer um certo nível de especialização, principalmente ao lidar com a VPC e o isolamento dos dados entre sub-redes públicas e privadas. Esse setup inicial pode ser complexo e

exigir um entendimento profundo da infraestrutura de nuvem, além de potencialmente aumentar o tempo de implementação do sistema.

- Dependência da AWS: A arquitetura depende fortemente dos serviços da AWS, como RDS, Cognito e CloudWatch, o que limita a flexibilidade para migração para outras plataformas de nuvem, caso seja necessário. Caso no futuro a escolha por outra provedora de nuvem se torne necessária, será necessário um esforço substancial para adaptar a arquitetura.

Em resumo, a arquitetura proposta atende aos principais requisitos de segurança, escalabilidade e disponibilidade, sendo adequada para o sistema de agendamento de consultas médicas. Contudo, a dependência de uma única plataforma de nuvem (AWS) e os custos associados à escalabilidade exigem monitoramento contínuo e consideração de estratégias de otimização a longo prazo.

8. Conclusão

A execução deste projeto arquitetural proporcionou importantes aprendizados sobre o planejamento, implementação e manutenção de uma solução em nuvem, especialmente em um contexto que exige alta disponibilidade, segurança e escalabilidade. A arquitetura proposta visa atender aos requisitos do sistema de agendamento de consultas médicas, considerando as necessidades de confiabilidade, eficiência e flexibilidade. A seguir, são apresentados os principais aprendizados, decisões arquiteturais e trade-offs envolvidos, assim como oportunidades de melhorias para versões futuras da arquitetura.

Lições Aprendidas

1. **Segurança versus Desempenho:** A implementação de uma VPC com sub-redes públicas e privadas garantiu segurança adicional ao isolar recursos internos e externos, mas também trouxe a necessidade de configurações complexas que podem afetar a latência e o tempo de resposta do sistema. A utilização de AWS Cognito para autenticação, embora essencial para a segurança, adiciona uma camada de processamento em cada requisição. Esse equilíbrio entre segurança e desempenho exigiu uma análise cuidadosa durante a definição da arquitetura.
2. **Modularidade e Preparação para Microsserviços:** A organização do sistema em módulos, como os domínios de médicos, pacientes e agendamentos, facilitou o desenvolvimento e a manutenção do sistema. Além disso, prepara o projeto para uma possível migração para uma arquitetura de microsserviços no futuro. Contudo, a granularidade inicial dessa estrutura exigiu um esforço significativo de planejamento e de gestão das dependências entre os módulos, o que pode aumentar a complexidade da manutenção à medida que o sistema cresce.

3. **Custo e Escalabilidade:** A adoção de serviços gerenciados da AWS, como o RDS PostgreSQL e o CloudWatch, proporcionou uma escalabilidade eficaz e monitoramento contínuo, fundamentais para o sistema de agendamento de consultas. Contudo, à medida que o uso desses serviços cresce, os custos operacionais aumentam consideravelmente. Foi necessário balancear as necessidades de escalabilidade com a sustentabilidade financeira da aplicação, especialmente em um cenário de crescimento rápido.
4. **Monitoramento e Manutenção Proativa**: A utilização do CloudWatch permitiu monitorar o sistema em tempo real, com a configuração de alertas para identificar problemas rapidamente e realizar manutenções preventivas. Este recurso foi essencial para garantir a disponibilidade e desempenho da aplicação. No entanto, ele também demandou uma configuração detalhada e uma análise contínua dos dados gerados, exigindo atenção constante para evitar falhas de performance e disponibilidade.

Possibilidades de Melhoria

Para versões futuras da arquitetura, algumas melhorias podem ser implementadas para otimizar o desempenho, a manutenibilidade e os custos:

1. **Introdução de Cache:** A inclusão de uma camada de cache, como Amazon ElastiCache, poderia melhorar significativamente o tempo de resposta em consultas frequentes, reduzindo a carga no banco de dados e melhorando a performance geral do sistema, especialmente durante picos de acesso ao sistema de agendamento.
2. **Automatização do Deploy:** A implementação de uma pipeline de CI/CD com AWS CodePipeline pode otimizar o processo de deploy, proporcionando atualizações mais rápidas e seguras. A automação também facilitaria o controle de versões e a integração contínua, contribuindo para uma maior agilidade no desenvolvimento.

3. **Balanceamento e Otimização de Custo**: À medida que o sistema cresce, será fundamental revisar periodicamente os recursos utilizados e ajustar a escalabilidade para manter um bom equilíbrio entre custo e desempenho. A utilização de instâncias spot para tarefas não críticas e a reavaliação dos níveis de escalabilidade podem ajudar a reduzir os custos operacionais, mantendo a eficiência da aplicação.
4. **Refinamento dos Módulos para Microsserviços**: Com a arquitetura já estruturada em módulos, uma evolução natural seria a refatoração para uma arquitetura de microsserviços completa. Isso permitiria a independência de desenvolvimento, deploy e escalabilidade de cada módulo, proporcionando maior flexibilidade à medida que o sistema se expande. A migração para microsserviços também traria vantagens em termos de manutenção, podendo tornar a aplicação mais modular e fácil de evoluir ao longo do tempo.

Considerações Finais

A arquitetura proposta atendeu aos principais requisitos do sistema de agendamento de consultas médicas, garantindo segurança, escalabilidade e disponibilidade. No entanto, como todo projeto de longo prazo, existem áreas que podem ser aprimoradas para garantir a evolução contínua da aplicação. O foco em modularidade e na preparação para uma migração futura para microsserviços são fatores chave para garantir que o sistema seja flexível e capaz de atender a futuras demandas. A implementação de melhorias como caching, automação de deploy e otimização de custos contribuirá para a evolução da solução, garantindo que o sistema permaneça eficiente e sustentável à medida que cresce.

Link do vídeo de apresentação: https://youtu.be/1ixHG24hwPE

Referências

Arquitetura de Soluções e Design de Software

- RICHARDS, M. *Fundamentals of Software Architecture: An Engineering Approach*. O'Reilly Media, 2020.
- FOWLER, M. *Patterns of Enterprise Application Architecture*. Addison-Wesley, 2002.
- EVANS, E. *Domain-Driven Design: Tackling Complexity in the Heart of Software*. Addison-Wesley, 2004.
- LARMAN, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design*. Prentice Hall, 2002.

Microsserviços e Arquitetura Monolítica

- NEWMAN, S. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, 2019.
- Amazon Web Services (AWS). *Documentação oficial da AWS sobre práticas de deployment e Spring Framework.* Disponível em: https://aws.amazon.com/architecture. Acesso em: [data de acesso].

Desenvolvimento em Nuvem e Microsserviços

- VILLAMIZAR, M., et al. *Infrastructure cost comparison of running web applications in the cloud using AWS Lambda and monolithic and microservice architectures.* International Journal of Cloud Computing, 2015.

Tópicos de Banco de Dados e Infraestrutura

- KLEPPMANN, M. *Designing Data-Intensive Applications*. O'Reilly Media, 2017.
- Documentação do PostgreSQL e AWS RDS. *Materiais sobre práticas de segurança e escalabilidade de bancos de dados relacionais na nuvem*. Disponível em: https://docs.aws.amazon.com. Acesso em: [data de acesso].

AWS e Serviços Gerenciados

- Amazon Web Services (AWS). *Documentação oficial da AWS sobre VPC, ALB/NLB, SES, Cognito, RDS*. Disponível em: https://docs.aws.amazon.com. Acesso em: [data de acesso].
- MAZONKA, O. *AWS for Solutions Architects: The complete beginner-to-advanced guide*. Packt Publishing, 2021.

Metodologia ATAM (Architecture Tradeoff Analysis Method)

- KAZMAN, R.; KLEIN, M.; CLEMENTS, P. *ATAM: Method for Architecture Evaluation*. Software Engineering Institute, Carnegie Mellon University, 2000.

Padrões e Práticas em Java e Spring Boot

- WALLS, C. *Spring in Action*. Manning Publications, 2018.
- BLOCH, J. *Effective Java*. Addison-Wesley, 2018.

User Interface (UI) e Experiência do Usuário (UX)

- NIELSEN, J.; BUDIU, R. *Mobile Usability*. New Riders, 2013.
- GARRETT, J. J. *The Elements of User Experience: User-Centered Design for the Web and Beyond*. New Riders, 2010.