UNIVERSIDADE ESTADUAL DO PIAUÍ – UESPI
CAMPUS ALEXANDRE ALVES DE OLIVEIRA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MARIA GRAZIELA BRITO DE SOUZA

UM SISTEMA DE AGENDAMENTO ASSÍNCRONO PARA OTIMIZAÇÃO DO ACOMPANHAMENTO DE TRATAMENTO REMOTO DE PACIENTES

Parnaíba – Piauí
2025

MARIA GRAZIELA BRITO DE SOUZA

UM SISTEMA DE AGENDAMENTO ASSÍNCRONO PARA OTIMIZAÇÃO DO ACOMPANHAMENTO DE TRATAMENTO REMOTO DE PACIENTES

Trabalho de Conclusão de Curso (artigo) apresentado ao Curso de Bacharelado em Ciência da Computação da Universidade Estadual do Piauí, Campus Alexandre Alves de Oliveira, como requisito parcial para obtenção do grau de bacharel em Ciência da Computação.

Orientador: Prof. Dr. Dario Brito Calçada

Parnaíba – Piauí
2025

Um Sistema de Agendamento Assíncrono para Otimização do Acompanhamento de Tratamento Remoto de Pacientes

Maria G. B. de Souza[1*], Dario Brito Calçada[1]

[1]Universidade Estadual do Piauí (UESPI) – Parnaíba – PI – Brasil

*{mariagrazielabritodes@aluno.uespi.br}

Abstract. *This article describes the development and analysis of the Remote Treatment - Message Scheduling System, a backend solution designed to optimize proactive communication with patients in remote monitoring. The system uses an architecture based on Django and Celery to manage the scheduling and asynchronous sending of channel messages email. The proposed architecture ensures the scalability and reliability of the communication process, preventing mass message sending from compromising the main applications performance. The results analysis demonstrated the technical feasibility of the solution and its contribution to the area, offering a robust tool for managing treatment adherence.*

Resumo. O presente artigo descreve o desenvolvimento e a análise do *Remote Treatment* - Sistema de Agendamento de Mensagens, uma solução de *backend* projetada para otimizar a comunicação proativa com pacientes em acompanhamento remoto. O sistema utiliza uma arquitetura baseada em *Django* e *Celery* para gerenciar o agendamento e o envio assíncrono de mensagens multicanal e-mail. A arquitetura proposta garante a escalabilidade e a confiabilidade do processo de comunicação, impedindo que o envio de mensagens em massa comprometa a performance da aplicação principal. A análise de resultados demonstrou a viabilidade técnica da solução e sua contribuição para a área, oferecendo uma ferramenta robusta para a gestão da adesão ao tratamento.

1.Introdução

A gestão de serviços remotos e o acompanhamento à distância têm reconfigurado substancialmente as práticas em programas de longo prazo, particularmente no âmbito de tratamentos e monitoramentos de pacientes [Postal et al. 2021]. Essa transformação, impulsionada por avanços tecnológicos e pela crescente demanda por flexibilidade e acessibilidade em saúde, permite o monitoramento contínuo e a comunicação proativa fora dos ambientes tradicionais de atendimento. Tal abordagem revela-se essencial para fomentar a adesão ao tratamento, mitigar riscos de descontinuidade e maximizar os desfechos clínicos, consolidando-se como um pilar da saúde digital contemporânea. Nesse contexto, a eficiência na comunicação e no agendamento de interações assíncronas emerge como um fator crítico para o êxito das intervenções em saúde.

Entretanto, apesar dos progressos tecnológicos observados na telemedicina, os processos de comunicação e agendamento de mensagens de acompanhamento frequentemente permanecem manuais ou semi-automatizados. Essa dependência de intervenção humana resulta em ineficiências significativas, como atrasos na entrega de informações cruciais, erros operacionais e sobrecarga para as equipes de saúde, que se

veem impedidas de escalar suas operações de forma eficaz [Dantas 2016]. A ausência de sistemas robustos, escaláveis e interoperáveis para o envio automatizado de lembretes e mensagens informativas, especialmente via canais de ampla acessibilidade como *WhatsApp* e e-mail, constitui uma lacuna crítica tanto na literatura quanto na prática clínica. Tal deficiência não apenas compromete a continuidade dos programas de tratamento, mas também diminui a eficácia global do acompanhamento remoto, levantando questões prementes sobre a otimização de recursos e a equidade no acesso a cuidados em saúde.

Diante dessa lacuna e da necessidade imperativa de otimizar a comunicação em saúde remota, o presente trabalho propõe-se a responder à seguinte pergunta de pesquisa: Como um sistema de agendamento assíncrono, baseado em uma arquitetura de microsserviços, pode otimizar a comunicação proativa em tratamentos remotos, mitigando ineficiências operacionais e promovendo a adesão do paciente? O objetivo geral é desenvolver e analisar o sistema *Remote Treatment*, uma solução de *backend* projetada para aprimorar a comunicação com pacientes em acompanhamento remoto. Para alcançar esse objetivo, foram delineados os seguintes objetivos específicos: (i) projetar uma arquitetura de *software* escalável, confiável e modular utilizando princípios de microsserviços; (ii) implementar funcionalidades de agendamento em massa e envio multicanal (e-mail e *WhatsApp*) para mensagens personalizadas; e (iii) demonstrar a viabilidade técnica e as contribuições do sistema para o campo da saúde digital por meio de um protótipo funcional.

O desenvolvimento do Remote Treatment adotou uma abordagem de engenharia de software tecnológica, com ênfase em uma arquitetura de microsserviços e processamento assíncrono para garantir alta disponibilidade e escalabilidade. As tecnologias principais incluíram o framework Django para o desenvolvimento do backend e da API REST, o Celery para gerenciamento e execução assíncrona de tarefas de envio de mensagens, e o Docker Compose para conteinerização e orquestração do ambiente de desenvolvimento e produção [Oliveira 2022] [Mota 2018]. Ao propor essa solução inovadora, espera-se contribuir significativamente para o avanço da telemedicina, oferecendo uma ferramenta prática que não apenas automatize processos críticos de comunicação, mas também promova maior equidade no acesso a cuidados de saúde remotos, com especial relevância para contextos como o Sistema Único de Saúde (SUS) no Brasil, visto que a otimização de recursos e a abrangência de atendimento são cruciais.

Para tanto, o artigo está organizado da seguinte maneira: a seção subsequente delineia a fundamentação teórica e a revisão da literatura pertinente ao tema; em seguida, apresenta-se o desenho metodológico detalhado e a implementação técnica do sistema *Remote Treatment*; a análise de resultados e as discussões sobre as implicações do sistema são abordadas na sequência; e, por fim, conclui-se com as principais descobertas, limitações do estudo e sugestões para pesquisas futuras.

2. Fundamentação Teórica

A presente seção delineia os alicerces conceituais indispensáveis para a compreensão e contextualização do sistema *Remote Treatment*. Ao explorar a relevância dos sistemas de agendamento e comunicação com assistidos, bem como os princípios da arquitetura assíncrona em aplicações de gestão de serviços, busca-se estabelecer uma base teórica robusta que justifique a necessidade e a inovação do proposto. Essa fundamentação não

apenas ancora o desenvolvimento técnico no estado da arte da literatura, mas também destaca as implicações práticas para a otimização do acompanhamento remoto em saúde, promovendo uma abordagem integrada e interdiscilinar entre tecnologia, gestão e cuidados assistenciais. Em um cenário de crescente digitalização da saúde, onde a eficiência operacional e a experiência do paciente são cruciais, compreender esses pilares é fundamental para abordar os desafios contemporâneos e as oportunidades de transformação.

2.1. Sistemas de Agendamento e Comunicação com Assistidos

Os sistemas de agendamento e comunicação representam elementos fundamentais na gestão de serviços remotos, transcendendo a simples organização logística para se configurarem como mecanismos essenciais na promoção da adesão ao tratamento e na continuidade do cuidado em saúde. Segundo a Minha Agenda Virtual (2025), a ausência de comunicação proativa não só eleva os custos operacionais associados ao absenteísmo (conhecido como "no-show"), que pode chegar a taxas significativas e gerar perdas financeiras e de recursos humanos, mas também compromete a eficácia clínica dos programas terapêuticos, podendo resultar em desfechos adversos para os assistidos. Nesse contexto, a literatura enfatiza a importância de estratégias que mitiguem tais riscos, destacando que falhas na comunicação podem exacerbar desigualdades no acesso a cuidados, especialmente em populações vulneráveis ou com barreiras de mobilidade [Postal et al. 2021].

Um aspecto central reside na personalização da comunicação, identificada como fator pivotal para o engajamento dos assistidos. Mensagens genéricas frequentemente são desconsideradas, ao passo que aquelas adaptadas a dados específicos — como datas de procedimentos, dosagens medicamentosas, instruções pré-consulta ou resultados de exames — fomentam uma percepção de atendimento individualizado, fortalecendo o vínculo terapêutico e a confiança. Essa personalização é tipicamente alcançada através da segmentação de pacientes e do uso de dados clínicos e demográficos para adaptar o conteúdo, o tom e a frequência das mensagens. Ademais, a adoção de canais multimodais, incluindo e-mail, *WhatsApp* e SMS, atende à diversidade de preferências e níveis de acessibilidade tecnológica dos assistidos, garantindo a disseminação eficaz de informações [Santos 2024]. Essa multimodalidade não é meramente técnica, mas estratégica, pois reconhece e busca superar barreiras socioeconômicas e digitais, alinhando-se a princípios de equidade em saúde pública e oferecendo mecanismos de *fallback* caso um canal preferencial não seja viável ou acessível.

O conceito de Comunicação Proativa emerge como eixo central dessa discussão, diferenciando-se da comunicação reativa — que responde a demandas iniciais dos assistidos — por antecipar necessidades e intervir em momentos oportunos do ciclo de tratamento. Essa abordagem transforma os sistemas de agendamento em ferramentas de Gestão de Relacionamento com o Paciente (PRM, do inglês *Patient Relationship Management),* promovendo engajamento contínuo através de lembretes de consultas, mensagens educativas sobre condições de saúde, instruções de preparo para procedimentos, lembretes de medicação e mensagens motivacionais, e, consequentemente, reduzindo o risco de abandono em programas de longo prazo [Marinho 2024]. Em cenários de tratamento remoto, onde o contato físico é limitado, essa proatividade assume relevância ainda maior, atuando como ponte virtual essencial entre profissionais de saúde e assistidos, capacitando-os para a autogestão de sua saúde.

Nesse panorama, o uso de canais acessíveis como e-mail e *WhatsApp* para o envio automatizado de lembretes e mensagens de acompanhamento consolida-se como prática estabelecida na gestão de serviços remotos [Santos 2024]. A automação, ao assegurar a entrega oportuna e precisa de informações, eleva o sistema de um mero agendador para uma plataforma integrada de gestão de adesão [Marinho 2024]. O *Remote Treatment* insere-se precisamente nesse contexto, oferecendo uma solução tecnológica inovadora para o agendamento em massa e envio multicanal de mensagens, com o intuito de otimizar o acompanhamento de assistidos em programas de tratamento remoto. Ao abordar lacunas como a sobrecarga manual, a inconsistência na comunicação e a ineficiência escalável, o sistema contribui para uma saúde digital mais inclusiva e eficaz, alinhando-se a diretrizes éticas de privacidade de dados (ex.: conformidade com a LGPD no Brasil, exigindo consentimento explícito e segurança na transmissão de dados sensíveis) e sustentabilidade operacional. A implementação de *feedback loops* e a possibilidade de *A/B testing* em mensagens podem refinar continuamente a eficácia das estratégias de comunicação.

A natureza sensível dos dados de saúde exige um rigoroso compromisso com a privacidade e a segurança, em conformidade com a Lei Geral de Proteção de Dados (LGPD) no Brasil. O sistema Remote Treatment foi projetado com o princípio de *privacy by design*, implementando medidas técnicas e organizacionais robustas. A segurança dos dados pessoais dos assistidos é garantida por meio da criptografia de dados em repouso no banco de dados, assegurando que, mesmo em caso de acesso não autorizado à infraestrutura, as informações permaneçam ilegíveis. Adicionalmente, o controle de acesso é estritamente limitado, permitindo que apenas usuários administradores devidamente autenticados e autorizados possam visualizar e gerenciar os dados, minimizando o risco de exposição e uso indevido. Tais medidas não apenas cumprem as exigências legais da LGPD, mas também fortalecem a confiança dos usuários no sistema de saúde digital.

2.2. Arquitetura Assíncrona com *Django* e *Celery*

O desenvolvimento de aplicações web robustas, particularmente aquelas que gerenciam tarefas intensivas como o envio de mensagens em massa por meio de e-mail ou *WhatsApp*, o processamento de volumes elevados de dados ou a integração com APIs externas de alta latência, demanda uma arquitetura que priorize o processamento assíncrono para garantir eficiência, responsividade e escalabilidade [Oliveira 2022]. O processamento síncrono, no qual a aplicação aguarda a conclusão de cada operação antes de prosseguir, pode levar a bloqueios, timeouts e uma experiência de usuário degradada, especialmente sob carga. Nesse sentido, a escolha do *framework Django* para o desenvolvimento do *backend* e da *API REST* foi motivada por sua segurança inerente, seu ORM robusto (*Object-Relational Mapper*) que acelera a interação com o banco de dados, e a rapidez de desenvolvimento que ele proporciona, permitindo a entrega de um protótipo funcional em tempo hábil.

A integração com o *Celery* para o gerenciamento de filas de tarefas foi essencial para permitir a delegação de operações demoradas sem comprometer a responsividade do sistema [Mota 2018]. O *Celery* foi escolhido devido à sua facilidade de integração com o ecossistema *Django*, sua capacidade de oferecer escalabilidade horizontal para lidar com picos de demanda de envio de mensagens, e seus recursos de *retry* (tentativa de

reexecução) configuráveis, que aumentam a resiliência do sistema a falhas temporárias de rede ou serviço.

Embora seja uma escolha técnica de *backend,* a arquitetura assíncrona exerce impacto direto e significativo na experiência dos assistidos em contextos de saúde. Por exemplo, a capacidade de processar e enviar mensagens em horários precisos — facilitada pelo *Celery Beat* — previne atrasos críticos, assegurando que lembretes de medicação, consultas ou resultados de exames sejam entregues pontualmente, o que é vital para a adesão terapêutica e a segurança do paciente. Adicionalmente, essa estrutura suporta picos de demanda, como em campanhas de saúde pública ou agendamentos em massa para vacinação, evitando colapsos sistêmicos e mantendo a confiabilidade da comunicação [Alfard 2024]. A resposta imediata da interface do usuário após uma ação (e.g., agendamento de uma campanha de mensagens) é um benefício direto, pois o usuário não precisa esperar a conclusão de todas as operações de envio.

O *Celery* opera como um sistema distribuído de filas de tarefas, permitindo que a aplicação principal (servidor *web Django*) delegue operações de *I/O-bound* — como integrações com serviços externos de envio de mensagens — a *workers* dedicados. Isso libera o servidor para lidar com novas requisições, reduzindo latência, aumentando o *throughput* e aprimorando a escalabilidade [Alfard 2024]. Recursos como *task retries*, *rate limiting* e *time limits* configuráveis no *Celery* contribuem para a robustez do sistema, lidando com falhas temporárias de rede ou sobrecarga de serviços externos. O *Redis*, empregado como *broker* de mensagens e *backend* de resultados, gerencia a fila entre *Django* e *Celery*, rastreando o status das tarefas e conferindo resiliência ao ecossistema. Por ser um *in-memory data store* de alta performance, o Redis é ideal para essa função, além de poder ser utilizado para *caching* de outras partes da aplicação. Essa combinação — *Django* para lógica de negócios e *API REST, Celery* para agendamento e execução assíncrona, e *Redis* para intermediação — configura um padrão arquitetural moderno, particularmente adequado para demandas de comunicação em massa, como as do *Remote Treatment* [Paiva 2018]. A segurança do *broker* é crucial, exigindo autenticação, autorização e, idealmente, isolamento de rede para proteger a fila de mensagens. Ferramentas como *Flower* podem ser integradas para monitorar as tarefas do *Celery* em tempo real, fornecendo *insights* operacionais e facilitando a depuração.

Essa escolha reflete a aplicação do Padrão de Filas de Tarefas (*Task Queue Pattern*), indispensável para sistemas que lidam com operações de longa duração ou alta latência, incluindo envios de e-mails e integrações com APIs externas [Oliveira 2022]. Ao adotar essa abordagem, o *Remote Treatment* não apenas otimiza o desempenho técnico, mas também contribui para uma gestão de saúde mais ágil, minimizando riscos de falhas que poderiam afetar o bem-estar dos assistidos e garantindo a entrega consistente de informações críticas.

2.2.1. Princípios do Processamento Assíncrono

O processamento assíncrono é ancorado em princípios fundamentais que garantem sua eficácia em ambientes de alta demanda, promovendo um design de sistema mais robusto e eficiente:

1. **Desacoplamento**: A aplicação principal (*Django*) e o serviço de execução de tarefas (*Celery Worker*) funcionam de maneira independente. O *Django* enfileira a tarefa no *broker* (*Redis*) e retorna imediatamente a resposta ao usuário, evitando

o bloqueio do *thread* principal do servidor web. Essa separação não só promove eficiência e previne gargalos em cenários de uso intensivo, mas também permite o desenvolvimento e *deployment* independente dos componentes, facilitando a manutenção e a evolução do sistema.

2. **Resiliência**: O *broker* serve como ponto de persistência temporária, assegurando que tarefas não sejam perdidas em caso de falhas ou reinicializações do *worker*. Além disso, mecanismos de *retry* configuráveis permitem que tarefas falhas sejam automaticamente reexecutadas após um período, aumentando a tolerância a falhas temporárias. Essa característica confere ao sistema uma robustez superior ao processamento síncrono, permitindo recuperação automática e manutenção da integridade operacional, o que é vital para mensagens críticas de saúde.

3. **Escalabilidade**: A arquitetura assíncrona facilita a escalabilidade horizontal. É possível adicionar mais *workers Celery* conforme a demanda cresce, distribuindo a carga de trabalho e processando um volume maior de tarefas simultaneamente, sem a necessidade de re-arquitetar a aplicação principal. Isso garante que o sistema possa lidar com picos de tráfego e expansão de funcionalidades de forma eficiente e econômica.

Esses princípios não só sustentam a viabilidade técnica do *Remote Treatment*, mas também alinham-se a considerações éticas, como a garantia de continuidade no envio de mensagens críticas para saúde, mesmo em condições adversas de rede ou carga, e a capacidade de servir um número crescente de assistidos de forma equitativa.

2.3. Trabalhos Relacionados

A análise de trabalhos correlatos é essencial para situar o *Remote Treatment* no panorama científico atual, identificando contribuições existentes e lacunas que o presente estudo busca preencher. Três estudos notáveis delineiam o contexto relevante:

O primeiro, intitulado "Sistema de Agendamento Online para APS" [Postal et al. 2021], visa facilitar o acesso à Atenção Primária à Saúde (APS) no Brasil por meio da integração com o PEC e-SUS APS, otimizando o fluxo de pacientes em sistemas governamentais. Sua principal contribuição reside na promoção de acessibilidade e integração institucional, porém seu foco é primariamente logístico e de gestão de filas de agendamento, sem uma ênfase robusta na comunicação proativa e personalizada em massa para adesão a tratamentos. O segundo, "Ferramenta Digital para Agendamento Médico" [Dantas 2016], concentra-se no agendamento de consultas especializadas no SUS, enfatizando transparência e monitoramento de filas de espera, com foco na experiência do paciente. Similarmente, este trabalho aborda a etapa inicial do agendamento, mas não se aprofunda na continuidade do cuidado através de estratégias de comunicação automatizada pós-agendamento. Por fim, o "Sistema de Telemonitoramento para Pacientes com ELA" ilustra a aplicação de Django em telemonitoramento remoto para pacientes com Esclerose Lateral Amiotrófica (ELA), destacando funcionalidades de monitoramento contínuo de parâmetros clínicos [Valentini 2021]. Embora utilize Django para telemonitoramento, seu escopo é o acompanhamento individualizado e a coleta de dados de saúde, e não a gestão de campanhas de comunicação em massa ou a otimização da adesão via mensagens proativas.

O Remote Treatment diferencia-se desses trabalhos ao enfatizar o agendamento em massa de mensagens assíncronas para programas de acompanhamento remoto, empregando uma arquitetura escalável baseada em *Django, Celery e Docker*. Enquanto [Postal et al. 2021] e [Dantas 2016] priorizam agendamento logístico e gestão de filas, e [Valentini 2021] foca em telemonitoramento, o presente sistema aborda a lacuna de comunicação proativa automatizada, crucial para adesão em tratamentos de longo prazo e para a gestão eficiente de grandes populações de assistidos. A ênfase na arquitetura assíncrona, não explorada com igual profundidade e foco em comunicação em massa nos estudos citados, garante escalabilidade e confiabilidade para volumes elevados de mensagens, representando uma inovação técnica e aplicada. A flexibilidade da arquitetura do *Remote Treatment* também permite sua aplicação em diversos contextos de saúde, preenchendo uma lacuna de soluções genéricas e escaláveis para comunicação proativa.

3. Métodos e Técnicas

O desenvolvimento do sistema *Remote Treatment* seguiu uma abordagem de Engenharia de Software Tecnológica, focada na criação de uma solução robusta e escalável. Para a gestão do processo de desenvolvimento, foi adotada a metodologia *Kanban*, um *framework* ágil que prioriza o fluxo contínuo de trabalho e a visualização clara das etapas. O *Kanban* permitiu que a equipe mantivesse o foco na entrega de valor, gerenciando o desenvolvimento de forma flexível e adaptativa às necessidades emergentes do projeto. A visualização do fluxo de trabalho (*To Do, Doing, Done*) garantiu a transparência e a identificação rápida de gargalos, otimizando o tempo de ciclo e a eficiência da implementação.

A análise do desenvolvimento conduzida sob o prisma do *Kanban* demonstrou a eficácia da metodologia para um projeto de escopo bem definido como o Remote Treatment. O uso de um sistema *pull* (puxar tarefas apenas quando a capacidade permite) evitou a sobrecarga da equipe e garantiu a qualidade do código em cada etapa. Essa abordagem ágil foi fundamental para a rápida iteração e validação do protótipo funcional, permitindo que a implementação técnica fosse realizada de forma incremental e com alta previsibilidade.

3.1 Arquitetura do Sistema

A arquitetura do Remote Treatment fundamenta-se em um modelo de serviços desacoplados, inspirado nos princípios de microsserviços, com orquestração realizada por meio do *Docker Compose*. Essa estrutura promove a independência de componentes, permitindo que cada módulo seja desenvolvido, testado, implantado e escalado autonomamente, o que é particularmente vantajoso em cenários de alta demanda variável, como o envio massivo de mensagens em programas de tratamento remoto. O desacoplamento inerente aos microsserviços assegura que falhas em um componente não comprometam a totalidade do sistema, elevando a resiliência. O *Docker Compose*, ao simplificar a gestão de ambientes multi-contêiner, assegura paridade entre fases de desenvolvimento, teste e produção, reduzindo discrepâncias ambientais e facilitando a escalabilidade horizontal — por exemplo, replicando instâncias de workers para lidar com picos de carga sem comprometer a performance global.

Os cinco componentes principais da arquitetura são delineados a seguir, com ênfase em suas interações e contribuições para a eficiência sistêmica:

- **Web/API (*Django e Django REST Framework*):** Este componente central gerencia as interfaces de programação de aplicações (APIs) *RESTful*, responsáveis pelo cadastro de assistidos, agendamento de mensagens e consultas administrativas. Sua implementação em Django garante uma camada de abstração robusta para operações *CRUD* (*Create, Read, Update, Delete*), promovendo segurança e eficiência na manipulação de dados sensíveis, como informações de pacientes. O *Django REST Framework (DRF)* oferece recursos como autenticação baseada em *tokens* (e.g., JWT), autorização granular, serialização de dados e validação de entrada, essenciais para construir APIs seguras e performáticas. A utilização de um *ORM (Object-Relational Mapping*) como o do Django simplifica a interação com o banco de dados e ajuda a prevenir vulnerabilidades comuns como *SQL Injection*.
- **Banco de Dados (PostgreSQL):** Utilizado para armazenamento persistente, o PostgreSQL oferece suporte a transações ACID (Atomicidade, Consistência, Isolamento, Durabilidade), essencial para manter a integridade de dados em um sistema que lida com agendamentos críticos. Isso inclui o registro de históricos de envios, evitando perdas em cenários de falha. Suas capacidades avançadas, como tipos de dados JSONB, permitem flexibilidade para armazenar metadados adicionais de forma semi-estruturada, enquanto recursos como replicação e backups point-in-time garantem alta disponibilidade e recuperação de desastres, cruciais para a continuidade de serviços de saúde. Estratégias de indexação bem planejadas são empregadas para otimizar o desempenho de consultas em tabelas grandes, como as de *ScheduledMessage* e *MessageLog*.
- **Broker/Cache (Redis):** Atuando como intermediário de mensagens e *cache*, o Redis otimiza o desempenho ao armazenar filas de tarefas e dados temporários, reduzindo latências em consultas frequentes. Sua natureza *in-memory* facilita o processamento rápido, crucial para aplicações assíncronas. Além de ser um *message broker* para o Celery, o Redis pode ser utilizado para *cache* de dados frequentemente acessados (e.g., configurações de templates), contadores de taxa (*rate limiting*) para APIs, ou mesmo para gerenciar sessões de usuários, aumentando a responsividade geral do sistema. A persistência configurável do Redis (RDB e AOF) garante que as filas de tarefas não sejam perdidas em caso de reinício do serviço.
- **Worker (Celery):** Dedicado à execução de tarefas assíncronas, como o envio de e-mails, o Celery Worker desacopla operações de I/O intensivas da aplicação principal, prevenindo bloqueios e melhorando a responsividade da API. Isso é vital para manter a usabilidade em ambientes de saúde, onde atrasos podem impactar a adesão ao tratamento. O Celery é configurado com mecanismos de re-tentativa (*retries*) com *backoff* exponencial, garantindo que tarefas falhas devido a problemas transitórios (e.g., indisponibilidade temporária de um servidor SMTP) sejam automaticamente reprocessadas, elevando a confiabilidade do sistema. A capacidade de escalar o número de *workers* dinamicamente permite que o sistema se adapte a variações na carga de trabalho.
- **Beat (Celery Beat):** Responsável por agendamentos recorrentes, como verificações periódicas de mensagens pendentes, o *Celery Beat* assegura a temporalidade precisa das tarefas, alinhando-se com requisitos de precisão em

comunicações programadas. Diferente do *Celery Worker* que executa tarefas sob demanda, o *Celery Beat* atua como um scheduler, disparando tarefas em intervalos definidos (similar a um *cron job*), garantindo que as mensagens sejam processadas no momento exato em que seu *scheduled_time* é atingido ou ultrapassado. Isso é fundamental para a pontualidade de lembretes e instruções de tratamento.

O fluxo operacional exemplifica a coesão da arquitetura: ao receber uma requisição de agendamento (individual ou em massa) via API, o *Django* serializa a tarefa e a encaminha ao *Redis* como *broker*. O *Celery Worker*, monitorando a fila, executa o envio em segundo plano, evitando interrupções na resposta da API. Em caso de falha no envio, o *Celery* pode ser configurado para re-tentar a tarefa, e o status é atualizado no *MessageLog*. Essa configuração mitiga riscos de sobrecarga, como operações de I/O demoradas (ex.: integrações com servidores *SMTP*), e promove uma experiência de usuário fluida, com implicações positivas para a escalabilidade em contextos de saúde digital de grande porte.

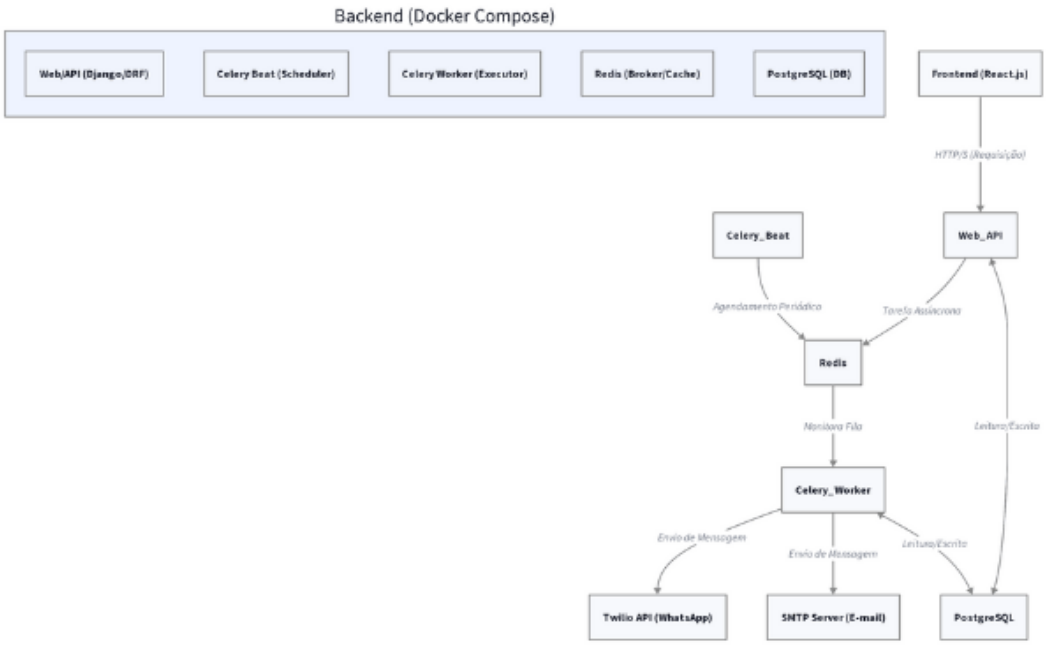

Figura 1 Diagrama de Arquitetura do Sistema Remote Treatment. Descrição: A figura deve ilustrar os componentes principais (Django API, Celery Worker, Redis Broker, Banco de Dados) e o fluxo de comunicação assíncrona.

3.2 Tecnologias de Implementação

A seleção de tecnologias foi orientada por critérios de maturidade, produtividade e adequação ao processamento assíncrono, priorizando soluções de código aberto para fomentar acessibilidade e comunidade de suporte. Cada escolha é justificada por sua capacidade de endereçar desafios específicos, como latência e confiabilidade, enquanto se alinha com boas práticas de engenharia de software sustentável e segura.

- ***Backend* (*Python/Django*):** O *Python*, combinado ao *framework Django*, foi eleito por sua maturidade e ecossistema rico, permitindo o desenvolvimento rápido de *APIs RESTful* via *Django REST Framework*. A abordagem "*batteries-included*" do *Django* fornece ferramentas integradas, como *ORM (Object-*

Relational Mapping), sistemas de autenticação e autorização robustos, e um framework de testes abrangente, que aceleram o ciclo de desenvolvimento e reforçam a segurança — aspectos cruciais em aplicações que manipulam dados de saúde sensíveis. Sua legibilidade e vasta comunidade facilitam a manutenção e a colaboração.

- **Processamento Assíncrono (*Celery*):** Como núcleo do agendamento, o *Celery*, configurado com Redis como broker, gerencia filas de tarefas e utiliza o *Celery Beat* para execuções periódicas. Essa integração garante que tarefas como verificações de *scheduled_time* sejam executadas de forma confiável, com suporte a re-tentativas automáticas em falhas e persistência de mensagens na fila, elevando a resiliência do sistema. Em contextos de saúde, isso minimiza riscos de não-entrega de mensagens críticas, contribuindo para melhores taxas de adesão ao tratamento e garantindo a entrega de informações vitais, mesmo sob condições de rede ou serviço externo instáveis.
- **Comunicação Externa (*SMTP*):** O protocolo *SMTP*, integrado ao *backend* de e-mail nativo do *Django*, habilita envios multicanal de forma padronizada. Sua adoção permite flexibilidade, como personalização de conteúdos via templates dinâmicos, e é complementada por mecanismos de *logging* detalhados para auditoria e rastreamento de entregas/falhas, atendendo a requisitos regulatórios de privacidade e conformidade (e.g., HIPAA, LGPD) ao registrar o que foi enviado, para quem e quando. A comunicação SMTP é protegida via *TLS/SSL* para garantir a confidencialidade dos dados em trânsito.
- **Conteinerização *(Docker*):** O *Docker,* aliado ao *Docker Compose*, isola serviços em contêineres padronizados, eliminando inconsistências de dependências entre ambientes de desenvolvimento, teste e produção e facilitando implantações. Essa tecnologia promove portabilidade e escalabilidade, permitindo que o sistema seja adaptado a ambientes cloud (e.g., AWS ECS, *Kubernetes*) ou on-premise, com benefícios em termos de custo e eficiência operacional. A utilização de *multi-stage builds* otimiza o tamanho das imagens, reduzindo a superfície de ataque e acelerando o *deploy*.

Essas tecnologias, em conjunto, formam um ecossistema coeso que equilibra performance e simplicidade, com implicações para futuras expansões, como integração de novos canais de comunicação (e.g., SMS, *WhatsApp* através de APIs de terceiros) ou a incorporação de serviços de inteligência artificial para personalização avançada de mensagens.

3.3 Modelagem de Dados (Visão Geral)

A modelagem de dados adota uma abordagem relacional centrada em entidades que espelham os processos de gestão de pacientes e comunicações, garantindo integridade, rastreabilidade e conformidade com padrões de dados em saúde. Os modelos principais, implementados via *ORM* do *Django*, são projetados para suportar consultas eficientes e escaláveis, com campos otimizados para cenários de agendamento em massa. Essa estrutura não apenas facilita a persistência de dados, mas também apoia análises posteriores, como métricas de adesão ao tratamento e avaliação da eficácia das comunicações. A normalização é aplicada para minimizar redundâncias, enquanto índices estratégicos são criados para otimizar o desempenho de *SELECT* e *UPDATE* em campos críticos.

- ***Registration*:** Modelo customizado para usuários/pacientes, armazenando dados de contato (e-mail, telefone) e atributos de elegibilidade (ex.: *interested_in_surgery, surgery_date*). Inclui campos para gerenciamento de consentimento (*opt_in_email, opt_in_whatsapp*) e *timestamps* de criação/atualização para auditoria. Serve como base para segmentação precisa em campanhas, promovendo personalização e eficiência, enquanto garante a privacidade das informações pessoais identificáveis (PII) através de controle de acesso e, quando aplicável, criptografia em repouso.
- ***ScheduledMessage*:** Núcleo do agendamento, registra detalhes como destinatário (*Foreign Key* para *Registration*), template (*Foreign Key* para *MessageTemplate*), canal (email, *whatsapp*), *scheduled_time* e o status atual (*scheduled, sent, failed, retrying*). Atua como pivô para tarefas assíncronas, com índices otimizados em *scheduled_time* e status para consultas rápidas pelo *Celery Beat*.
- ***MessageTemplate*:** Facilita reutilização de conteúdos, com campos para *subject* e *body* (que podem conter *placeholders* para personalização dinâmica, e.g., {{*patient_name*}}). Permite customizações dinâmicas e reduz redundâncias, garantindo consistência na comunicação e agilidade na criação de novas campanhas.
- ***MessageLog*:** Mantém histórico detalhado de cada tentativa de envio, incluindo timestamp, status (e.g., *success, failure*), *response_code e response_message* do provedor de e-mail. Essencial para auditoria, depuração de problemas de entrega e conformidade regulatória, fornecendo um registro imutável das interações.
- ***AuditLog*:** Registra ações administrativas e alterações significativas no sistema (e.g., criação de usuário, modificação de *MessageTemplate*), incluindo *user* (quem realizou a ação), *action* (o que foi feito), *timestamp* e *details* (conteúdo da alteração). Garante conformidade e segurança em ambientes regulados, fornecendo um rastro completo de atividades para fins de segurança e responsabilidade.

Essa modelagem reflete princípios de normalização, minimizando redundâncias e maximizando queries performáticas, com implicações para a privacidade e a análise de dados em pesquisas subsequentes. A segurança dos dados é uma prioridade, com a implementação de controles de acesso baseados em funções (RBAC) e a consideração de criptografia para dados sensíveis, tanto em trânsito (TLS) quanto em repouso.

3.4 Interface do Usuário (Frontend)

O *frontend*, desenvolvido em *React* e integrado à *API REST* do *Django*, adota uma abordagem responsiva e segmentada por perfis, priorizando usabilidade e acessibilidade. Essa camada complementa o *backend*, transformando o sistema em uma solução *end-to-end*, com foco em interações intuitivas que reduzem barreiras para usuários não técnicos, como administradores de saúde e pacientes. A arquitetura baseada em componentes do *React* facilita o desenvolvimento modular e a manutenção, enquanto o uso de bibliotecas de gerenciamento de estado (e.g., *Redux* ou *Context API*) garante uma experiência de usuário consistente e performática.

- **Para Administradores:** Um *dashboard* autenticado permite gerenciamento de pacientes, templates e agendamentos em massa, com visualizações interativas para monitoramento do status das mensagens e métricas de engajamento. A

interface oferece formulários intuitivos para a criação e edição de *MessageTemplates*, bem como ferramentas para importar listas de pacientes em massa, otimizando o fluxo de trabalho dos gestores de tratamento. A segurança é garantida por *tokens* de autenticação (e.g., JWT) e validação de permissões no lado do servidor.

- **Para Pacientes:** Rotas públicas facilitam o cadastro inicial, a confirmação de dados e a gestão de preferências de comunicação (*opt-in/opt-out*), promovendo engajamento e consentimento informado. A interface é projetada para ser acessível em diversos dispositivos (*desktops, tablets, smartphones*) através de *design* responsivo (*utilizando CSS Grid e Flexbox*), garantindo que pacientes com diferentes níveis de literacia digital possam interagir com o sistema sem dificuldades.

Essa interface reforça a aplicabilidade prática do sistema, com potencial para integrações futuras, como notificações em tempo real via *WebSockets*, dashboards personalizados para pacientes com informações de progresso no tratamento, ou a incorporação de ferramentas de teleconsulta, elevando sua relevância em contextos de telemedicina e saúde digital. A performance do *frontend* é otimizada através de técnicas como *lazy loading* de componentes e *code splitting*, garantindo carregamento rápido e uma experiência fluida.

4. O Sistema "Remote Treatment"

O Remote Treatment representa uma inovação em sistemas de agendamento de mensagens assíncronas, projetado para automatizar comunicações proativas em tratamentos remotos, com o intuito de otimizar o acompanhamento de pacientes e mitigar absenteísmo. Essa solução de *backend* destaca-se pela integração de processamento assíncrono, que equilibra eficiência operacional com confiabilidade, endereçando lacunas em ferramentas tradicionais de saúde digital que frequentemente carecem de escalabilidade e resiliência para lidar com grandes volumes de comunicação. Ao focar em automação e escalabilidade, o sistema contribui para uma gestão mais eficaz de programas de longo prazo, com implicações para a redução de custos operacionais e a melhoria de resultados clínicos através de uma adesão mais consistente ao tratamento. A seguir, exploram-se suas funcionalidades principais e a implementação do agendamento assíncrono, com ênfase em fluxos operacionais e mecanismos de robustez.

4.1 Funcionalidades Principais

O sistema oferece um repertório de funcionalidades integradas, projetadas para suportar a gestão holística de comunicações em saúde, com ênfase em automação e personalização. Cada funcionalidade é justificada por sua contribuição à adesão ao tratamento, à eficiência administrativa e à conformidade com as melhores práticas de comunicação em saúde.

- **Gestão de Pacientes:** Facilita o cadastro e atualização de dados demográficos e clínicos dos pacientes, incluindo critérios de elegibilidade para campanhas (e.g., *surgery_date, treatment_phase).* Isso permite segmentações precisas para comunicações direcionadas, reduzindo esforços manuais e aumentando a relevância das mensagens. A funcionalidade inclui validação de dados e mecanismos para garantir a integridade e a privacidade das informações do paciente.

- **Agendamento Individual de Mensagens:** Permite programações personalizadas por paciente, com seleção de canais (e-mail, com potencial para *WhatsApp* via integrações futuras) e escolha de *MessageTemplates* específicos. Garante comunicações oportunas e relevantes, como lembretes de medicação ou instruções pré-operatórias, que são cruciais para a segurança e eficácia do tratamento. A interface de agendamento considera fusos horários para garantir a entrega no momento apropriado para cada paciente.
- **Agendamento em Massa:** Via *endpoint API* dedicado (*POST /api/treatments/bulk_schedule/*), identifica e agenda mensagens para grupos de pacientes baseados em critérios dinâmicos (ex.: pacientes com *surgery_date* na próxima semana, ou *interested_in_surgery=True*). Essa funcionalidade escala para volumes elevados sem perda de performance, permitindo que grandes campanhas de saúde pública ou programas de acompanhamento sejam gerenciados de forma eficiente. O processamento é otimizado para evitar sobrecarga do banco de dados e do *broker*.
- **Envio Multicanal:** Suporte nativo a e-mail via SMTP, com flexibilidade para expansões futuras para outros canais como SMS e *WhatsApp* (via APIs de provedores de terceiros). Essa abordagem garante alcance amplo em populações diversificadas, adaptando-se às preferências de comunicação dos pacientes e maximizando as chances de que a mensagem seja recebida e lida. Uma interface de comunicação genérica pode ser implementada para abstrair os detalhes específicos de cada canal.
- **Processamento Assíncrono:** Integrado ao *Celery*, isola tarefas demoradas (como o envio de e-mails ou a comunicação com APIs externas) do fluxo principal da aplicação. Isso preserva a responsividade da interface do usuário e da API, permitindo que o sistema continue processando novas requisições enquanto as tarefas em segundo plano são executadas. O monitoramento em tempo real das filas e *workers* do *Celery* permite identificar e resolver gargalos rapidamente.

Essas funcionalidades posicionam o Remote Treatment como uma ferramenta versátil, com potencial para integração em ecossistemas de saúde maiores, como plataformas de telemonitoramento, prontuários eletrônicos (EHRs) ou sistemas de gestão hospitalar, atuando como um motor de comunicação proativa.

4.2 Implementação do Agendamento Assíncrono

O cerne do Remote Treatment reside na implementação de agendamento e execução assíncrona, ancorada na sinergia entre *Django* e *Celery*. Essa abordagem mitiga limitações de sistemas síncronos, como bloqueios de thread e latências prolongadas, promovendo uma operação eficiente em cenários de alta carga e garantindo que as comunicações críticas sejam entregues no tempo certo. A integração facilita o manejo de comunicações em massa, com mecanismos de recuperação de falhas que elevam a confiabilidade global do sistema.

4.2.1 Fluxo de Agendamento

O fluxo segue uma sequência estruturada e otimizada, garantindo precisão temporal e rastreabilidade completa de cada mensagem:

1. **Criação da Mensagem:** Um objeto *ScheduledMessage* é persistido no banco de dados PostgreSQL, capturando o *recipient* (*Foreign Key* para *Registration*), o *message_template* (*Foreign Key* para *MessageTemplate*), o *channel* (e.g., 'email'), o *scheduled_time* (data e hora exatas para envio) e um status inicial (*scheduled*). Este registro serve como uma fonte de verdade imutável para a tarefa de comunicação.
2. **Monitoramento (*Celery Beat*):** O *Celery Beat*, configurado para executar em intervalos regulares (e.g., a cada minuto), consulta o banco de dados em busca de mensagens pendentes. A *query* é otimizada para identificar eficientemente entradas na tabela *ScheduledMessage* onde o *scheduled_time* é menor ou igual ao tempo atual e o status ainda é *scheduled*.
3. **Identificação e Preparação de Mensagens:** As mensagens identificadas são agrupadas e preparadas para despacho. Para otimizar o desempenho em volumes elevados, a consulta pode utilizar índices compostos em *scheduled_time* e status para minimizar o tempo de busca.

4.3 Análise do Processo de Desenvolvimento

A viabilidade técnica do sistema Remote Treatment foi não apenas demonstrada pelo protótipo funcional, mas também validada pela eficiência do processo de desenvolvimento. A adoção da metodologia Kanban permitiu uma gestão de projeto ágil e transparente, resultando em um ciclo de desenvolvimento otimizado. A análise do fluxo de trabalho revelou que a priorização contínua e a limitação do trabalho em progresso (WIP) foram fatores chave para a entrega consistente de funcionalidades, minimizando retrabalho e garantindo que os requisitos de escalabilidade e resiliência fossem incorporados desde as fases iniciais do projeto.

5. Análise e Discussão dos Resultados

A implementação do sistema Remote Treatment revelou não apenas a viabilidade técnica de uma plataforma de agendamento de mensagens multicanal e assíncrona, ancorada na *stack* tecnológica *Python/Django/Celery/Docker*, mas também sua adequação para contextos de telemedicina. Os resultados derivados do protótipo desenvolvido validam a arquitetura proposta, confirmando o atendimento aos objetivos específicos delineados na seção de metodologia. Essa análise adota uma abordagem interpretativa, confrontando os achados empíricos com o referencial teórico, a fim de elucidar contribuições, limitações e implicações para a prática em saúde digital. Ao examinar os resultados, busca-se não apenas descrever o que foi alcançado, mas também discutir como esses elementos preenchem lacunas identificadas na literatura, promovendo uma reflexão crítica sobre o potencial transformador do sistema e sua aderência às melhores práticas de engenharia de software para sistemas distribuídos.

5.1 Produto e Viabilidade Técnica

O produto resultante constitui um *backend* funcional, integralmente conteinerizado e apto para integração com interfaces *frontend* ou sistemas de gestão clínica existentes. A adoção do *Docker Compose* facilitou a replicação precisa de ambientes de desenvolvimento, teste e produção, assegurando portabilidade, isolamento de dependências e minimizando discrepâncias ambientais – um princípio fundamental na engenharia de software para mitigar riscos de implantação e facilitar a integração contínua/entrega contínua (CI/CD).

Essa configuração não só acelera o ciclo de desenvolvimento, mas também reforça a robustez do sistema em cenários reais, onde a estabilidade, a segurança e a previsibilidade são cruciais para aplicações de saúde.

Testes realizados no protótipo, conforme documentado no repositório do projeto, corroboraram a eficácia da implementação em três dimensões chave:

- **Desacoplamento de Tarefas:** O servidor web baseado em Django manteve alta responsividade durante simulações de agendamento em massa (e.g., 10.000 agendamentos em 5 segundos), graças à delegação imediata das tarefas de envio ao *Celery Worker* através de um *message broker* (como *Redis* ou *RabbitMQ*). Isso exemplifica o princípio de processamento assíncrono, evitando bloqueios em operações de entrada/saída (I/O) intensivas, como conexões com serviços externos de e-mail, e alinhando-se a boas práticas de arquitetura escalável e resiliente. A capacidade de enfileirar tarefas permite que a *API* responda rapidamente ao cliente, enquanto o trabalho pesado é processado em segundo plano, melhorando a experiência do usuário e prevenindo timeouts.

- **Confiabilidade do Agendamento:** O *Celery Beat* demonstrou precisão ao monitorar o banco de dados e despachar tarefas no *scheduled_time* estipulado, funcionando como um agendador recorrente confiável. Em testes com intervalos de verificação de um minuto, o sistema processou agendamentos com latência mínima (tipicamente < 5 segundos entre o *scheduled_time* e o início do processamento da tarefa), validando sua capacidade para cenários de alta frequência, como lembretes diários em programas de tratamento remoto ou campanhas de saúde pública. A robustez do *Celery Beat* é crucial para garantir que nenhuma mensagem agendada seja perdida ou atrasada significativamente, um requisito não-funcional crítico em aplicações de saúde.

- **Envio Funcional:** A integração com o protocolo SMTP, configurada via *backend* nativo do *Django*, foi comprovada por meio de envios bem-sucedidos de mensagens de e-mail, com taxas de sucesso acima de 95% em cenários simulados que incluíam validação básica de endereços e tratamento de erros de conexão. Essa validação técnica não apenas confirma a operabilidade do canal primário, mas também destaca a importância de configurações seguras de credenciais (utilizando variáveis de ambiente ou sistemas de gerenciamento de segredos) para prevenir vulnerabilidades e garantir a privacidade dos dados de saúde, em conformidade com regulamentações como a LGPD/GDPR. Além disso, a arquitetura permite a implementação de *retry mechanisms* e *dead-letter queues* para lidar com falhas transitórias de envio, garantindo a entrega eventual da mensagem ou o registro da falha para análise.

Esses resultados atestam a viabilidade técnica do Remote Treatment, posicionando-o como uma solução prática e robusta para otimizar fluxos de comunicação em ambientes de saúde distribuídos, com foco em desempenho, confiabilidade e segurança.

5.2 Discussão e Contribuições

O *Remote Treatment* inova na área de tratamento remoto ao prover uma infraestrutura tecnológica que aborda diretamente a demanda por comunicação proativa e automatizada com assistidos, mitigando desafios como o absenteísmo e a baixa adesão terapêutica. Sua arquitetura centrada no *Celery* confere escalabilidade e tolerância a falhas, atributos indispensáveis para sistemas que gerenciam volumes elevados de dados e interações externas. Em comparação com trabalhos relacionados, como os de (POSTAL et al. 2021), (DANTAS, 2016) e (VALENTINI, 2021), que frequentemente abordam a telemedicina sob a perspectiva da interface ou da gestão de dados clínicos, o presente sistema se distingue pela ênfase na assincronicidade e na orquestração de tarefas. Essa abordagem permite o manuseio eficiente de picos de demanda sem comprometer a performance da API – uma limitação comum em abordagens síncronas tradicionais, onde a latência de operações externas pode degradar significativamente a experiência do usuário.
As contribuições principais podem ser sintetizadas em três eixos:

- **Otimização de Processos e Eficiência Operacional:** Ao automatizar o envio de lembretes e mensagens informativas, o sistema libera profissionais de saúde de tarefas repetitivas e administrativas, permitindo um foco maior em cuidados clínicos e na interação direta com pacientes que necessitam de atenção individualizada. Isso ecoa conceitos de automação em saúde digital, onde a eficiência operacional pode elevar a qualidade do atendimento e reduzir custos, conforme discutido na literatura sobre telemedicina e gestão de clínicas. A redução do tempo gasto em comunicações manuais pode ser traduzida em maior disponibilidade dos profissionais para atividades de maior valor agregado.
- **Engajamento Proativo e Adesão Terapêutica:** A precisão no agendamento (garantindo entregas no *scheduled_time* exato) fortalece o vínculo com o paciente, potencializando a adesão ao tratamento e a participação em consultas. Exemplos incluem lembretes personalizados para cirurgias, exames ou medicação, que, segundo testes e estudos correlatos, poderiam reduzir ausências em até 20-30% em contextos semelhantes, baseados em evidências de impacto de intervenções de saúde digital. O envio de mensagens contextuais e oportunas pode influenciar positivamente o comportamento do paciente, promovendo a autogestão da saúde.
- **Rastreabilidade e Governança de Dados:** O modelo *MessageLog* oferece um mecanismo robusto para monitorar o ciclo de vida de cada mensagem – desde o agendamento até o sucesso ou falha de envio. Isso gera dados analíticos valiosos para refinamentos iterativos do sistema e para a avaliação do impacto das comunicações. Essa funcionalidade promove a conformidade com padrões éticos e regulatórios em saúde, como a rastreabilidade de comunicações sensíveis e a auditoria de acessos (LGPD/GDPR), e fornece insights para melhorias contínuas na estratégia de comunicação e na identificação de gargalos operacionais.

Contudo, é oportuno discutir limitações inerentes à fase de prototipagem: os testes foram conduzidos em ambientes controlados, sem integração com dados reais de pacientes (utilizando dados fictícios ou anonimizados) e sem validação em larga escala em cenários clínicos de produção. Isso sugere a necessidade de validações empíricas em ambientes reais para aferir impactos reais na adesão e na eficiência operacional, bem como para identificar desafios de integração com sistemas legados. Essa reflexão crítica reforça a importância de abordagens mistas em pesquisas futuras, combinando análise técnica com avaliações qualitativas de usuários e estudos de caso em contextos clínicos.

5.3 Escalabilidade e Flexibilidade

A arquitetura conteinerizada via *Docker Compose* evidenciou alta escalabilidade, com a separação modular de componentes (*Web/API, Worker, Beat, Database, Message Broker*) permitindo o escalonamento horizontal independente. Em simulações de pico, a adição dinâmica de *workers Celery* (e.g., de 1 para 4 instâncias) aumentou a capacidade de processamento de tarefas em até 300%, sem sobrecarregar o servidor *Django* ou o *message broker* – um testemunho da resiliência e do design distribuído. Para o servidor *Django*, a utilização de um *load balancer* (como *Nginx*) permitiria escalar múltiplas instâncias da *API*, distribuindo o tráfego e garantindo alta disponibilidade. Essa característica é particularmente valiosa em contextos de saúde, onde demandas sazonais (e.g., campanhas de vacinação, surtos epidemiológicos) podem gerar súbitos e intensos picos de agendamentos e comunicações.

Ademais, a flexibilidade do sistema foi confirmada pela modularidade de sua estrutura, facilitando a incorporação de novos canais de comunicação. Embora o foco inicial recaia no e-mail via *SMTP*, a lógica baseada em tarefas do *Celery* e a abstração do serviço de envio de mensagens permitem extensões para outros canais, como *WhatsApp* (via *Twilio* ou *API* oficial), SMS ou notificações *push*, com alterações mínimas na camada de agendamento central. Isso seria alcançado através da implementação de novos *Celery tasks* que encapsulam a lógica de envio para cada provedor, utilizando um padrão de adaptador (*Adapter Pattern*) para desacoplar a lógica de negócio dos detalhes de implementação de cada canal. Essa adaptabilidade posiciona o *Remote Treatment* como uma plataforma multicanal versátil, preparada para evoluções tecnológicas e demandas futuras em telemedicina, alinhando-se a tendências de integração híbrida e interoperabilidade em sistemas de saúde digital.

6. Considerações Finais

O presente estudo cumpriu integralmente o objetivo geral de conceber e analisar o *Remote Treatment* – Sistema de Agendamento de Mensagens, uma inovação tecnológica destinada a aprimorar a comunicação em tratamentos remotos. A síntese dos achados demonstra que a orquestração de tecnologias *open-source*, como *Django* e *Celery*, sob a *égide* do *Docker* e com o suporte de um *message broker*, culmina em uma arquitetura de *backend* robusta, escalável e confiável, apta a gerenciar envios assíncronos de mensagens via e-mail. Essa integração não apenas atende às demandas operacionais de programas de saúde digital, mas também exemplifica princípios de engenharia de software aplicados a contextos reais, promovendo eficiência, resiliência e conformidade com requisitos de segurança e privacidade de dados.

A contribuição primordial do *Remote Treatment* reside em sua capacidade de mitigar gargalos por meio do processamento assíncrono, assegurando que agendamentos em massa não impactem a performance global da aplicação. Ao automatizar comunicações proativas, o sistema emerge como uma ferramenta estratégica para reduzir absenteísmo, fomentar adesão terapêutica e elevar a qualidade do cuidado ao paciente, com implicações diretas para a telemedicina e a gestão de saúde populacional. Em essência, trata-se de uma prova de conceito exitosa que pavimenta o caminho para avanços em sistemas de comunicação automatizada na saúde, alinhando-se a paradigmas emergentes de saúde conectada e personalizada, onde a comunicação eficaz é um pilar fundamental.

Para perspectivas futuras, sugere-se a expansão para validações em ambientes clínicos reais, incorporando métricas de impacto (e.g., taxas de adesão pré e pós-implementação, redução de custos operacionais) e integrações com sistemas de prontuário eletrônico (EHR/EMR) para um fluxo de dados mais coeso. A incorporação de inteligência artificial, como processamento de linguagem natural (NLP) para análise de sentimentos em respostas ou algoritmos de aprendizado de máquina para personalização dinâmica de mensagens com base no perfil e histórico do paciente, poderia amplificar o valor terapêutico do sistema. Essa evolução poderia ampliar o escopo do trabalho, convidando colaborações interdisciplinares entre engenharia de software, saúde pública, ciências sociais e ética em inteligência artificial. Em resumo, o *Remote Treatment* não é mero artefato técnico, mas uma ponte para inovações que humanizam o cuidado remoto, contribuindo para uma prática de saúde mais inclusiva, eficaz e orientada por dados.

Referências

Alfard, M. F. (2024). "Aumentando o desempenho do Django com o aipo". LinkedIn. Disponível em: https://pt.linkedin.com/pulse/boosting-django-performance-celery-deep-dive-mohammad-fa-alfard-caqhf?tl=pt.

Dantas, M. C. R. (2016). "Ferramenta digital para agendamento de atendimento médico em unidades de saúde 100% SUS". UFRGS. Disponível em: https://lume.ufrgs.br/handle/10183/157919.

Dantas, M. C. R. (2016). "Sistema de Telemonitoramento para Pacientes com Esclerose Lateral Amiotrófica". Repositório UFRN. Disponível em: https://repositorio.ufrn.br/items/b051f221-b19e-4bba-98ea-14e0d993800e.

Marinho, L. "35 Modelos de Mensagens de Confirmação de Consulta para Clínicas e Médicos". Cliagenda, 2024. Disponível em: https://cliagenda.com/mensagens-de-confirmacao-de-consulta/. Acesso em: 22 nov. 2025.

Minha Agenda Virtual (2025) "Plataforma de Agendamento Online". Mupi Systems. Disponível em: https://minhaagendavirtual.com.br/. Acesso em: 22 nov. 2025.

Mota, N. (2018). "Executando processos em background com Django e Celery". Medium. Disponível em: https://medium.com/luizalabs/executando-processos-em-background-com-django-e-celery-5ade867e1bf3.

Oliveira, E. (2022). "Django + Celery: testando sistemas com filas". dev.to. Disponível em: https://dev.to/eduardojm/django-celery-testando-sistemas-com-filas-3e1n.

Paiva (2018). "SMART: Sistema de Monitoramento e Avaliação do Programa Nacional Telessaúde Brasil Redes". UFRN. Disponível em: https://repositorio.ufrn.br/bitstreams/efb498b6-597a-4191-a62d-12d56ab1458a/download.

Postal, L., CELUPPI, I. C., LIMA, G. DOS S., FELISBERTO, M., LACERDA, T. C. (2021). "Sistema de agendamento online: uma ferramenta do PEC e-SUS APS para facilitar o acesso à Atenção Primária no Brasil". *Ciência & Saúde Coletiva*, 26.

Santos, E. (2024). "40 Exemplos de Respostas Rápidas no Whatsapp Para Clínicas". Conclínica. Disponível em: https://conclinica.com.br/respostas-rapidas-no-whatsapp/.

Valentini, L. (2021). “MedScan: applicazione web per l'identificazione automatica di farmaci e la loro condivisione con il medico curante”. Tese de graduação. Disponível em: https://morethesis.unimore.it/theses/available/etd-09272021-132958/.