

HISIUS: SOLUÇÃO DIGITAL PARA GERENCIAMENTO DE FILAS DE PRONTO-ATENDIMENTO EM AMBIENTES HOSPITALARES

Centro Paula Souza

Etec Prof.^a Ermelinda Giannini Teixeira – Santana de Parnaíba - SP

Técnico em Desenvolvimento de Sistema Integrado ao Ensino Médio

Gabriel Estevão Gonçalves Amaro

Giuliane Ferreira Rodrigues

Guilherme de Carvalho Guedes

Gustavo Cavalcanti Leite

João Pedro Costa Souza

Prof. Orientador: Cristiano Teixeira

RESUMO

A superlotação hospitalar continua sendo um dos desafios mais urgentes dos sistemas de saúde contemporâneos, pois impacta diretamente a experiência dos pacientes, a carga de trabalho das equipes médicas e a qualidade geral dos serviços prestados. Este artigo apresenta o HISIUS, uma plataforma digital inteligente desenvolvida para aprimorar a transparência na comunicação com os pacientes sobre os tempos de espera e otimizar a gestão de filas em ambientes hospitalares. O sistema permite que os pacientes acompanhem remotamente sua posição estimada na fila, reduzindo a ansiedade e minimizando conflitos decorrentes da superlotação. Além disso, oferece um recurso de pré-cadastro online para evitar aglomerações presenciais e disponibiliza painéis analíticos que auxiliam gestores de saúde na identificação de picos de demanda e gargalos operacionais. Ao integrar inovação tecnológica à administração hospitalar, o HISIUS contribui para aprimorar a eficiência do fluxo de atendimentos, elevar a satisfação dos pacientes e apoiar uma alocação mais eficaz de recursos institucionais.

Palavras-chave: superlotação hospitalar, gestão de filas, tecnologia em saúde, experiência do paciente, sistemas digitais de saúde

ABSTRACT

Hospital overcrowding remains one of the most pressing challenges in contemporary healthcare systems, as it directly affects patient experience, healthcare staff workload, and the overall quality of medical services. This paper presents *HISIUS*, an intelligent digital platform developed to enhance transparency in patient communication regarding waiting times and to optimize queue management in hospital environments. The system enables patients to remotely track their estimated position in line, thereby reducing anxiety and minimizing conflicts associated with excessive waiting. Additionally, it provides an online pre-registration feature to prevent on-site crowding and offers analytical dashboards that assist healthcare managers in identifying demand peaks and operational bottlenecks.

Keywords: hospital overcrowding, queue management, healthcare technology, patient experience, digital health systems.experience; Healthcare optimization.

LISTA DE ILUSTRAÇÕES

FIGURAS

Figura 1 - Realizar Registro na App	14
Figura 2 - Mover Paciente da Triagem para Fila de Atendimento	15
Figura A.1 - Requisitos Funcionais HISIUS	
Figura A.2 - Requisitos Não Funcionais HISIUS	
Figura A.3 - Diagrama de Caso de Uso	
Figura A.4 - DER (Diagrama Entidade-Relacionamento)	
Figura A.5 - MER (Modelo Entidade-Relacionamento)	
Figura A.6 - Diagrama de Classes	
Figura B.1 - Realizar Login no App	
Figura B.2 - Entrar com Código do Hospital	
Figura B.3 - Fila de Espera da Triagem	
Figura B.4 - Fila Para Atendimento Médico	
Figura B.5 - Editar Dados Pessoais	
Figura C.1 - Adicionar a Sala que o Paciente Será Atendido	
Figura C.2 - Copiar Dados do Paciente	
Figura C.3 - Ver informações do Funcionário	
Figura C.4 - Ver Administradores	
Figura C.5 - Ver Relatório	
Figura C.6 – Adicionar Funcionário na Equipe	

SUMÁRIO

1. INTRODUCÃO	5
2. REFERENCIAL TEÓRICO.....	6
2.1. Superlotação Hospitalar	6
2.2. Gestão de Filas e Classificação de Risco	7
2.3. Documentação e Requisito de Software	8
2.3.1. Requisitos Funcionais e Não Funcionais	8
2.3.2. Diagrama de Casos de Uso	9
2.3.3. Modelagem de Dados e Diagramas	9
3. METODOLOGIA	10
4. DESENVOLVIMENTO DO SISTEMA.....	11
4.1 Arquitetura e Tecnologias Utilizadas	11
4.2 Aplicação da Modelagem do Sistema	12
4.3 Processo de Desenvolvimento	13
4.4 Interface do Sistema e Funcionalidades	14
4.4.1. Paciente	14
4.4.2 Gestores/Funcionários	15
5. CONCLUSÃO E CONSIDERAÇÕES FINAIS	16
6. REFERÊNCIAS.....	

1. INTRODUÇÃO

De acordo com Anne Petter (2012), a superlotação em serviços hospitalares ocorre quando há uma alta demanda na qual excede a capacidade de atendimento que, por sua vez, afeta diretamente a percepção dos usuários sobre a qualidade do serviço prestado, demandando estratégias para organização e mitigação desse evento. Em ambientes hospitalares, esse problema assume uma proporção ainda mais relevante, pois, além de comprometer a experiência do paciente e o desempenho dos profissionais de saúde, também atinge o direito fundamental à saúde, como prevê o Artigo 196, inciso III da CF.

Neste cenário, as filas de espera tornam-se uma maneira de organização do fluxo de atendimento que geralmente é utilizada em estabelecimentos comerciais, bancos, hospitalares e outros. Em um ambiente hospitalar, a superlotação é ainda pior, podendo causar ansiedade, conflitos e problemas operacionais (PETTER, 2012).

Conforme é apresentado por Plytiuk Buzzi, pacientes, profissionais de saúde e organizações mantenedoras têm enfrentados obstáculos decorrentes da qualidade dos serviços prestadores. No ano de 2017 até 2019, foi realizada uma pesquisa no Hospital Geral Alfa, localizado no estado do Pernambuco, foi utilizado o Protocolo de Triagem de Manchester como método de observação e análise. Constatou-se um total de 167.743 registros de atendimento sem diferença estatística sobre o tempo de espera para pacientes que se classificam entre emergência, urgente e muito urgente. Por tanto, estes dados caem sobre a equipe hospitalar e sobrecarregam a estrutura médica e de enfermagem, impedindo que casos classificados como muito urgente não recebam prioridade em seu atendimento.

Diante desse cenário, o problema de pesquisa que orienta este trabalho é: quais soluções podem ser oferecidas a fim de melhorar a transparência na comunicação com o paciente sobre o tempo de espera e a gestão de filas hospitalares.

O objetivo geral deste Trabalho de Conclusão de Curso é oferecer soluções a fim de melhorar a transparência na comunicação com o paciente sobre tempo de espera e a gestão das filas hospitalares e tendo como objetivos específicos visar reduzir a ansiedade e conflitos gerados pela superlotação, também evitar que o pré-cadastro seja realizado no local, auxiliando a organização do hospital, de modo que

seja possível encontrar pontos para uma maior atenção.

Acredita-se que, como solução, a implementação de uma plataforma digital na qual ofereça transparência sobre a posição do paciente na fila de espera, permitindo de forma remota evitar aglomerações, propicie um formulário para o pré-cadastro online e geração de relatórios para identificar picos de demanda e ajudar na alocação de recurso possa contribuir significativamente para a otimização do atendimento hospitalar e a melhoria da comunicação entre hospital e paciente.possa

Este trabalho fundamenta-se na lacuna existente entre os avanços tecnológicos disponíveis e sua aplicação na melhoria do fluxo de pacientes em hospitais públicos. Ao desenvolver um sistema integrado que permita o monitoramento em tempo real da posição da fila de atendimento; possibilidade de espera remota; pré-cadastro digital e geração de relatórios. Tem-se como objetivo a melhoria da experiência dos pacientes e uma ferramenta para instituições de saúde.

2. REFERENCIAL TÉORICO

O referencial teórico tem como objetivo apresentar os fundamentos conceituais que sustentam o desenvolvimento do sistema HISIUS, abordando as principais causas e consequências da superlotação hospitalar, bem como os métodos e ferramentas utilizados na organização do atendimento. Além disso, esta seção busca relacionar o uso de tecnologias digitais como uma alternativa para otimizar os fluxos de trabalho e melhorar a comunicação entre pacientes e instituições de saúde.

2.1. Superlotação Hospitalar

A superlotação hospitalar é um dos maiores desafios enfrentados pelos sistemas de saúde públicos e privados. De acordo com Petter (2012), esse fenômeno ocorre quando o número de pacientes que necessitam de atendimento ultrapassa a capacidade física e operacional do hospital. Essa situação impacta diretamente a qualidade dos serviços prestados e compromete o bem-estar de pacientes e profissionais.

Conforme Buzzi e Plytiuk (2011), a superlotação resulta, em grande parte, da falta de planejamento e de estratégias adequadas de gestão de fluxo. A ausência de um controle eficiente causa longos períodos de espera, aumento do estresse nas

equipes médicas e maior risco de falhas no atendimento. Já Silva (2015) destaca que a administração do fluxo de pessoas dentro do ambiente hospitalar é fundamental para garantir eficiência, segurança e humanização no processo de cuidado.

Esses estudos reforçam a necessidade de adotar novas soluções de monitoramento e controle do atendimento. Nesse sentido, sistemas informatizados podem atuar como ferramentas de apoio à gestão, proporcionando dados em tempo real e auxiliando na tomada de decisões. O HISIUS surge nesse contexto como uma proposta tecnológica que visa reduzir os impactos da superlotação e melhorar a experiência do paciente durante o processo de espera.

2.2. Gestão de Filas e Classificação de Risco

A gestão de filas é um elemento essencial na administração hospitalar, pois garante que os atendimentos sejam realizados de forma organizada e justa, respeitando o nível de urgência de cada paciente. Segundo Anjos (2022), a ausência de um controle automatizado das filas contribui para atrasos, falhas de comunicação e insatisfação dos usuários.

Para reduzir esses problemas, muitos hospitais adotam o Protocolo de Manchester, um método que define a ordem de prioridade dos atendimentos com base na gravidade dos sintomas apresentados pelos pacientes. De acordo com o Ministério da Saúde (BRASIL, 2014), esse protocolo utiliza uma escala de cores que indica o tempo máximo de espera para cada caso: o vermelho representa emergências que exigem atendimento imediato; o laranja corresponde a situações muito urgentes, nas quais o atendimento deve ocorrer em até dez minutos; o amarelo identifica casos urgentes, que podem aguardar até trinta minutos; o verde é destinado a casos pouco urgentes, com tempo de espera de até cento e vinte minutos; e o azul é reservado para situações não urgentes, em que o atendimento pode ser realizado em até duzentos e quarenta minutos.

Essa padronização permite que os profissionais priorizem casos críticos e distribuam de forma mais eficiente os recursos humanos e estruturais, garantindo que o atendimento ocorra dentro de limites seguros de tempo. De acordo com Mussi e Krudycz (2024), o uso de protocolos de triagem e ferramentas tecnológicas de monitoramento contribui para maior eficiência e transparência nos serviços de pronto-

atendimento. No entanto, a falta de integração entre as etapas do processo e a ausência de sistemas acessíveis ainda dificultam a comunicação com o paciente.

Nesse contexto, o sistema HISIUS surge como uma solução tecnológica capaz de integrar o controle de filas e a comunicação com os usuários em tempo real. A plataforma permite que os pacientes acompanhem sua posição e o tempo estimado de espera diretamente pelo aplicativo, enquanto os profissionais de saúde e gestores têm acesso a relatórios automáticos e indicadores de fluxo. Dessa forma, o HISIUS busca unir tecnologia e gestão hospitalar para oferecer um atendimento mais eficiente, ágil e humanizado.

2.3 Documentação e Requisitos de Software

O processo de desenvolvimento do sistema HISIUS foi orientado pelos princípios da Engenharia de Software e estruturado a partir da elaboração de documentos técnicos que descrevem as funcionalidades, restrições e objetivos do sistema. A documentação de requisitos é essencial, pois garante que o produto final atenda às necessidades reais dos usuários e mantenha a consistência entre as etapas de análise, modelagem e implementação.

O HISIUS é um sistema híbrido, composto por uma aplicação web voltada para o uso interno de profissionais de saúde e gestores hospitalares, e um aplicativo móvel destinado aos pacientes. O sistema foi desenvolvido em React e React Native, utilizando Node.js com Express para o backend, MySQL como banco de dados relacional e Turborepo como arquitetura de repositório monolítico distribuído. Essa combinação de tecnologias permite a integração entre diferentes plataformas, garantindo desempenho, escalabilidade e facilidade de manutenção.

2.3.1. Requisitos Funcionais e Não Funcionais

Os requisitos funcionais representam as ações que o sistema deve executar, enquanto os requisitos não funcionais tratam das qualidades desejáveis, como desempenho, segurança e usabilidade. Entre os principais requisitos funcionais definidos no projeto, destacam-se: o cadastro de usuários e pacientes; o login com diferentes níveis de acesso (paciente, enfermeiro e administrador); a inserção de pacientes na fila conforme a classificação de risco; a visualização em tempo real do tempo médio de espera; a geração de relatórios gerenciais; e o envio de notificações

automáticas quando a vez do paciente estiver próxima.

(As tabelas completas de requisitos podem ser consultadas no Anexo A.)

Os requisitos não funcionais garantem o bom desempenho e a estabilidade do sistema. O HISIUS foi projetado para suportar até 500 acessos simultâneos sem perda significativa de desempenho, apresentar tempo de resposta inferior a dois segundos em operações comuns e manter compatibilidade com navegadores modernos e dispositivos móveis. As senhas são armazenadas de forma criptografada com bcrypt, e todas as ações críticas são registradas em logs de segurança, assegurando rastreabilidade e confiabilidade.

(As tabelas completas de requisitos podem ser consultadas no Anexo A.)

2.3.2. Diagramas de Caso de Uso

O diagrama de caso de uso ilustra as principais interações entre os usuários e o sistema. Foram definidos três perfis principais: paciente, profissional de saúde e administrador. O paciente pode realizar o pré-cadastro, acompanhar sua posição na fila e receber notificações sobre o tempo de espera estimado. O profissional de saúde tem acesso às filas de atendimento e pode registrar novos pacientes, atualizar informações e classificar riscos conforme o Protocolo de Manchester. Já o administrador é responsável por configurar permissões, gerar relatórios e supervisionar o funcionamento geral da plataforma. Essa modelagem garante clareza na definição das responsabilidades de cada ator e facilita futuras expansões do sistema.

(O diagrama completo de Casos de Uso podem ser consultadas no Anexo A.)

2.3.3. Modelagem de Dados e Diagramas

A modelagem do banco de dados foi realizada por meio dos diagramas MER (Modelo Entidade-Relacionamento) e DER (Diagrama Entidade-Relacionamento), que representam as tabelas, atributos e relações entre os dados. O modelo contempla entidades como Paciente, Profissional, Atendimento, Fila, Classificação de Risco e Notificação, permitindo que o fluxo de informações seja estruturado de forma lógica e eficiente.

(As tabelas completas de MER/DER podem ser consultadas no Anexo A.)

O Diagrama de Classes complementa a modelagem, descrevendo a estrutura interna do sistema e as relações entre seus componentes. Ele representa as classes principais, como Usuário, Fila, Atendimento e Notificação, além dos métodos responsáveis pelas operações do sistema, como autenticação, registro de log e atualização de status do paciente.

(O Diagrama de Classe completo pode ser consultadas no Anexo A.)

A integração entre os diagramas garante uma visão completa do funcionamento do HISIUS, desde o fluxo de interação até a estrutura de armazenamento de dados, permitindo maior controle e rastreabilidade sobre os processos de atendimento.

3. METODOLOGIA

A metodologia utilizada neste trabalho se baseia em uma combinação de abordagens teóricas e práticas voltadas ao desenvolvimento de um sistema digital para auxiliar na gestão de filas hospitalares. O projeto HISIUS foi desenvolvido de forma colaborativa entre os integrantes do grupo, orientado pelo professor Cristiano Teixeira, e busca aplicar os conhecimentos adquiridos no curso técnico em Desenvolvimento de Sistemas para solucionar um problema real observado em hospitais públicos: a falta de transparência e eficiência no controle de filas de espera.

O estudo feito pela nossa equipe, pode ser classificado como uma pesquisa aplicada, pois tem como principal objetivo propor uma solução prática para um problema existente no âmbito hospitalar, utilizando a tecnologia como meio para melhorar a comunicação entre hospital e paciente. Além disso, também se identifica como pesquisa exploratória e descritiva, já que investiga o problema da superlotação e descreve o funcionamento atual das filas hospitalares, buscando compreender suas causas e consequências antes de propor uma solução. Quanto à abordagem, a pesquisa combina aspectos qualitativos e quantitativos, analisando tanto as percepções de pacientes e profissionais da saúde pelos artigos científicos observados, quanto dados numéricos obtidos por meio de formulários.

O procedimento técnico usado é o levantamento de dados, realizado por meio de um questionário aplicado ao nosso público-alvo e aos profissionais de saúde, totalizando 74 respostas válidas. Essa coleta permitiu identificar os principais fatores que geram insatisfação no atendimento e que reforçam a necessidade de uma

ferramenta tecnológica para melhorar a gestão das filas e a transparência no tempo de espera.

A primeira etapa da pesquisa consistiu em um estudo bibliográfico, baseado em autores como Petter (2012), Silva (2015), Buzzi e Plytiuk (2011) e documentos do Ministério da Saúde (2014). Essa fase teve o objetivo de compreender o fenômeno da superlotação hospitalar e analisar métodos de priorização, como o Protocolo de Manchester, que estabelece faixas de tempo máximo de espera de acordo com a gravidade dos casos.

Logo em seguida, foi realizada a nossa modelagem do sistema, assim, sendo feita a criação dos diagramas de caso de uso, classes, MER e DER, bem como a elaboração da tabela de requisitos funcionais e não funcionais. Essa modelagem do sistema, serviu como base para estruturar as funcionalidades do HISIUS e orientar a fase de desenvolvimento.

Por fim, iniciou-se o desenvolvimento do sistema HISIUS, que está sendo implementado nas plataformas React (versão web) e React Native (versão mobile), com Node.js e Express no backend e MySQL como banco de dados. O projeto adota a arquitetura Turborepo, que possibilita o gerenciamento conjunto de aplicações web e mobile em um mesmo repositório, otimizando o fluxo de desenvolvimento.

4. DESENVOLVIMENTO DO SISTEMA

Esta seção descreve as etapas de construção do sistema HISIUS, apresentando o processo de implementação e integração das tecnologias definidas na fase de modelagem. O desenvolvimento foi orientado pelos requisitos e diagramas elaborados anteriormente, que serviram como base para estruturar as funcionalidades e a lógica do sistema.

4.1. Arquitetura e Tecnologias Utilizadas

O HISIUS foi criado como uma solução híbrida composta por um sistema web e um aplicativo móvel, com o objetivo de atender de forma diferenciada os dois principais perfis de usuários:

- Profissionais da saúde e administradores hospitalares, que acessam o sistema por meio da interface web para gerenciar filas, realizar cadastros e consultar

relatórios;

- Pacientes, que utilizam o aplicativo mobile para acompanhar o andamento do atendimento e obter informações sobre sua posição aproximada na fila.

A aplicação foi desenvolvida utilizando o framework React para a interface web e o React Native para o aplicativo móvel, ambos conectados a uma API construída em Node.js com o framework Express. Essa API é responsável por gerenciar a comunicação entre o front-end e o banco de dados, armazenando e recuperando informações em MySQL, um sistema de gerenciamento relacional que é amplamente utilizado em aplicações web.

O projeto foi estruturado sobre a arquitetura Turborepo, que permite a integração entre as versões web e mobile em um mesmo repositório, simplificando a manutenção e o compartilhamento de código entre os dois ambientes. Essa estrutura também contribui para uma maior escalabilidade do sistema, permitindo futuras expansões e novas funcionalidades sem necessidade de reestruturação completa.

A escolha dessas tecnologias visou equilibrar desempenho, portabilidade e facilidade de manutenção, além de aproveitar o amplo suporte da comunidade e a compatibilidade entre bibliotecas utilizadas em React e Node.js.

4.2. Aplicação da Modelagem do Sistema

A modelagem, apresentada anteriormente no Referencial Teórico (Seção 2.3), orientou toda a construção do HISIUS. Os diagramas de caso de uso, classes, MER e DER foram fundamentais para entendermos as interações entre usuários, processos e dados do sistema.

Com base neles, foram definidos os principais módulos:

- Módulo de autenticação, responsável pelo login de pacientes e funcionários com diferentes níveis de acesso;
- Módulo de fila, que controla o cadastro, a posição e o tempo estimado de espera dos pacientes;
- Módulo administrativo, que permite ao hospital visualizar relatórios e configurar notificações;
- Módulo de relatórios, voltado à análise de picos de demanda e fluxos de

atendimento.

Os requisitos funcionais e não funcionais, também apresentados anteriormente, serviram como guia de validação das funcionalidades implementadas, garantindo que o sistema atendesse às demandas de desempenho, segurança e usabilidade estabelecidas.

4.3. Processo de Desenvolvimento

O processo de desenvolvimento foi dividido em etapas sequenciais, garantindo uma progressão estruturada entre a fase de criação e a implementação prática.

As etapas seguiram a seguinte ordem:

- - Planejamento e levantamento de requisitos – análise do problema, definição dos requisitos e objetivos do sistema;
- - Modelagem – elaboração dos diagramas e estruturação lógica do sistema;
- Implementação inicial – desenvolvimento dos módulos principais (login, cadastro, controle de fila);
- - Integração com o banco de dados – criação das tabelas e rotas de comunicação entre front-end e backend;
- - Desenvolvimento do aplicativo mobile – adaptação das principais funcionalidades para a versão React Native;
- - Testes e validações – verificação de funcionamento, responsividade e desempenho (etapa ainda em andamento).

O sistema HISIUS está em sua fase final de desenvolvimento, com as versões web e mobile já integradas e funcionando corretamente em ambiente local. As funcionalidades principais já foram implementadas, demonstrando a viabilidade da nossa solução proposta, e o sistema se encontra em um estágio avançado de integração e testes.

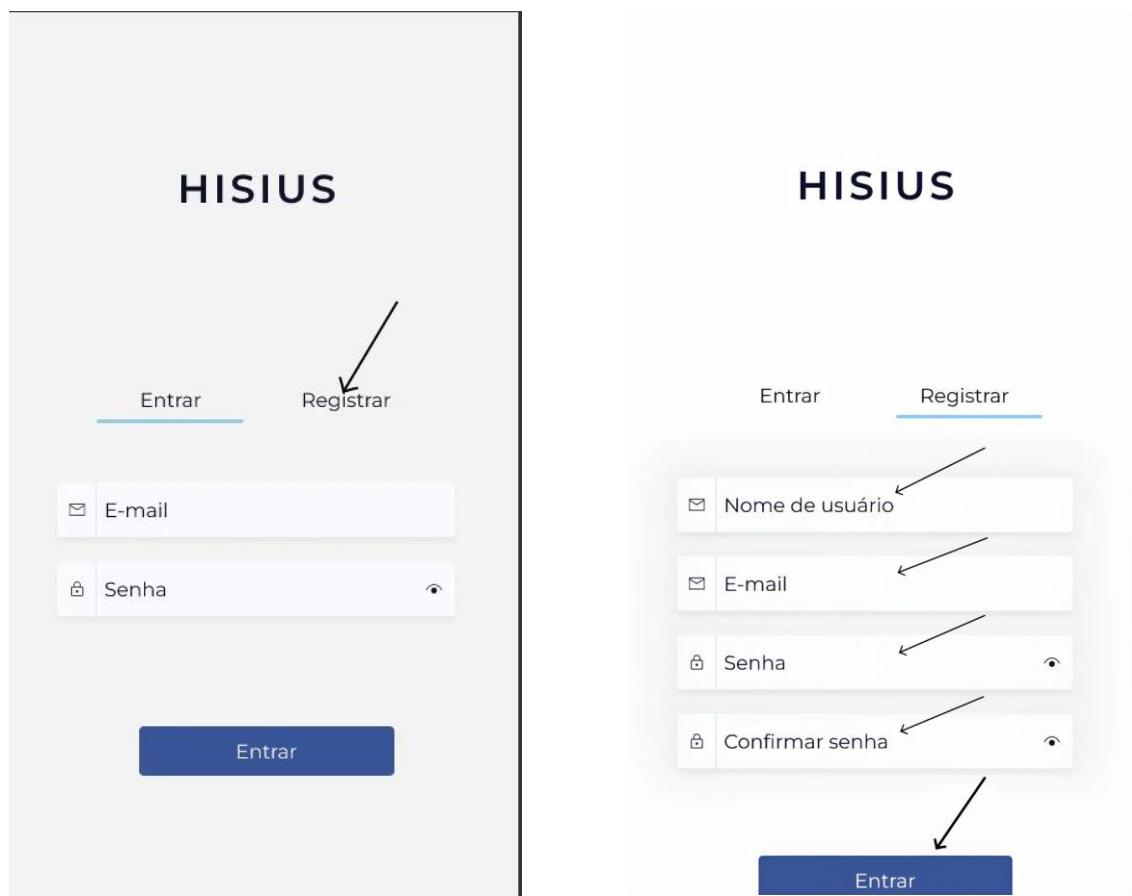
4.4. Interface do Sistema e Funcionalidades

A seguir, são apresentadas as principais telas desenvolvidas do sistema HISIUS, demonstrando o funcionamento das suas interfaces web e mobile. As imagens representam o estado atual do sistema e suas funcionalidades principais.

4.4.1 Paciente

Esta seção tem como objetivo ilustrar e descrever o processo de prototipagem das telas disponíveis para os usuários do tipo paciente no projeto HISIUS.

Figura 1 – Realizar Registro na app



Fonte: Autoria Própria (2025)

A Figura 1 ilustra o comportamento da aplicação durante o processo de Realizar Registro no App. O processo se inicia ao clicar no botão "Registrar". Em seguida, será redirecionado à tela de Registro, onde os dados solicitados são preenchidos. Ao clicar em "Entrar", o usuário é redirecionado para a tela inicial do app.

Devido a alta quantidade de telas, as demais figuras sobre as telas do sistema estarão disponíveis no Anexo B

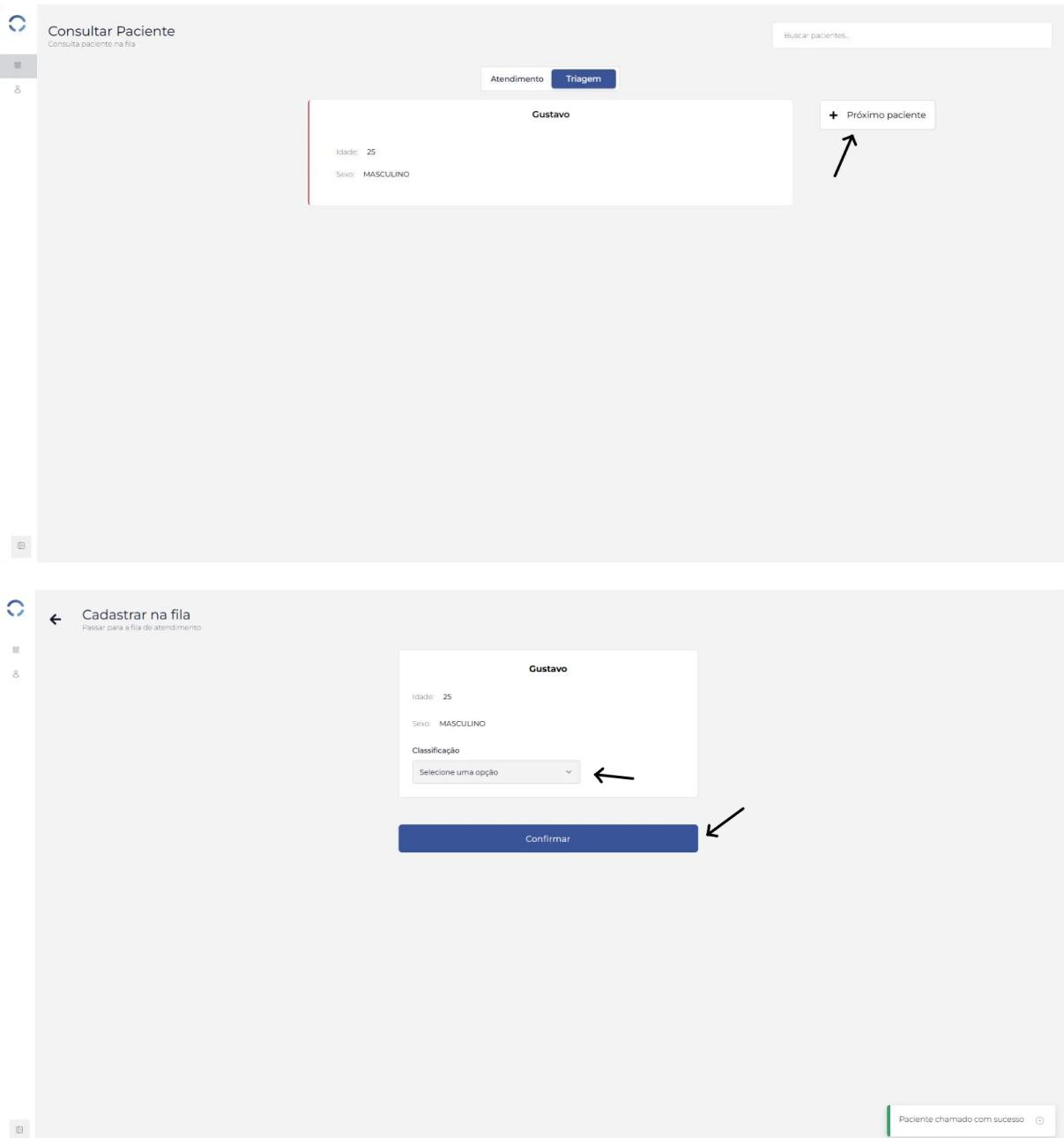
(As telas podem ser visualizados no Anexo B.)

4.4.2 Gestores/Funcionários

Esta seção tem como objetivo ilustrar e descrever o processo de prototipagem das

telas disponíveis para os usuários do tipo Gestores ou Funcionários no projeto HISIUS.

Figura 2 – Mover Paciente da Triagem para Fila de Atendimento



Fonte: Autoria Própria (2025)

A figura 2 ilustra o comportamento da aplicação durante o processo de mover paciente da triagem para fila de atendimento. O processo se inicia após clicarmos em próximo paciente. Em seguida, será necessário colocar a classificação de risco do paciente. Ao clicar em "Confirmar", será salvo os dados e o paciente será movido para a fila de atendimento médico.

Devido a alta quantidade de telas, as demais figuras sobre as telas do sistema estarão disponíveis no Anexo C

5. CONCLUSÃO E CONSIDERAÇÕES FINAIS

O desenvolvimento do sistema HISIUS possibilitou compreender, de forma prática, como a tecnologia pode contribuir para a solução de problemas reais no contexto hospitalar, especialmente em relação à superlotação e à falta de transparência no gerenciamento das filas de atendimento.

Durante a elaboração deste trabalho, foi possível aplicar os conhecimentos adquiridos ao longo do curso técnico em Desenvolvimento de Sistemas, unindo teoria e prática em um projeto voltado ao bem-estar da população e à melhoria da eficiência dos serviços de saúde. O HISIUS demonstrou ser uma ferramenta viável para integrar pacientes, profissionais e gestores, oferecendo uma experiência mais organizada e humanizada no atendimento hospitalar.

A pesquisa realizada confirmou a relevância da proposta, evidenciando que a principal causa de insatisfação entre os pacientes está relacionada à ausência de informações claras sobre o tempo de espera. O sistema desenvolvido busca justamente sanar essa lacuna, permitindo o acompanhamento em tempo real da posição na fila e a realização de pré-cadastros digitais, reduzindo aglomerações e otimizando o fluxo de atendimento.

O nosso projeto HISIUS já demonstrou seu potencial para ser implantado em ambientes hospitalares reais, com a maioria das funcionalidades em operação total. Ajustes de usabilidade e testes de segurança continuam sendo realizados para garantir a melhor experiência ao usuário e a estabilidade do sistema. Sua arquitetura permite a expansão para novas funcionalidades, como a integração com sistemas de prontuário eletrônico e a geração de relatórios estatísticos.

Conclui-se, portanto, que a implementação de soluções tecnológicas como o HISIUS pode representar um passo importante para modernizar a gestão hospitalar, promovendo maior transparência, eficiência e qualidade no atendimento ao paciente. O projeto reflete não apenas a aplicação de técnicas de desenvolvimento de software, mas também o compromisso social em buscar inovações que contribuam para o aprimoramento dos serviços públicos de saúde no Brasil.

6. REFERÊNCIAS

ANJOS, Geanderson Ribeiro. **Sistema de gerenciamento de filas para atendimento presencial.** Uruaçu: Campos Uruaçu, 2022.

PETTER, Anne Karine Fritsch. **A Superlotação do Serviço de Emergência Hospitalar.** Três de Maio: Universidade Federal de Santa Maria, 2012.

JUSBRASIL. Art. 196 da **Constituição Federal de 1988.**

SILVA, Mariana Ferreira. **Gestão de Fluxos de Pessoas num Serviço Hospitalar.** Minho, 2015. 83. Dissertação, (Mestrado em Engenharia Industrial) – Escola de Engenharia, Universidade do Minho.

MORAIS, Matheus Araujo. **Otimizando o Tempo do Atendimento Hospitalar: Uma Solução para Filas em Hospitais.** João Pessoa: Universidade Federal da Paraíba, 2023.

BUZZI, D.; PLYTIUK, C. F. **Pensamento enxuto e sistemas de saúde: um estudo da aplicabilidade de conceitos e ferramentas lean em contexto hospitalar.** Revista Qualidade Emergente, v. 2, n. 2, p. 18-38, 2011.

MUSSI, Fabricio Baron; KRUDYCZ, Laís Carolini. **Atendimentos Priorizados e Redução de Tempo de Espera:** Proposta de Melhorias em um Pronto Atendimento. Foz do Iguaçu: PUC, 2024.

BRASIL. **Classificação de Risco 12.8.22.** [S.I.], 2014

BRASIL. **Constituição (1988).** Constituição da República Federativa do Brasil de 1988. Brasília, DF: Presidência da República, 1988.

MORAIS, Larissa Oliveira de; ALMEIDA, Vítor Santos; SANTOS, Letícia Moraes dos. **O protocolo de Manchester como ferramenta de melhora dos serviços de emergência.** Revista da Faculdade de Ciências Médicas de Sorocaba, v. 23, n. 2, p. 87–93, 2021

Disponível em: <https://docs.bvsalud.org/biblioref/2021/07/1281663/o-protocolo-de-manchester.pdf>

GRUPO PORTUGUÊS DE TRIAGEM. **Protocolo de Triagem de Manchester.** Lisboa: GTP, 2023.

Disponível em: <https://www.grupoportuguestriagem.pt/grupo-portugues-triagem/protocolo-triagem-manchester/>

ANEXOS

ANEXO A – LINK DO GITHUB

<https://github.com/GustavoCavalcantii/Hisius>

ANEXO B – Requisitos Funcionais e Não Funcionais; Diagrama Caso de Uso e DER/MER

Figura B.1 – Requisitos Não Funcionais HISIUS

Identificador	Descrição	Prioridade	Depende de
RF 01	O sistema deve permitir o cadastro dos usuários	Alta	
Identificador	Descrição	Prioridade	Depende de
RF 02	O sistema deve permitir o login com diferentes níveis de acesso(paciente e enfermeiro).	Alta	RF01
Identificador	Descrição	Prioridade	Depende de
RF 03	O sistema deve permitir que o funcionário cadastre os pacientes na fila de atendimento de acordo com a classificação de risco	Alta	RF02
Identificador	Descrição	Prioridade	Depende de
RF 04	O sistema deve permitir que a equipe médica visualize os pacientes e seus dados nas filas de espera	Alta	RF03
Identificador	Descrição	Prioridade	Depende de
RF 05	O sistema deve permitir que o paciente visualize o tempo médio de espera, atualizado dinamicamente com base na fila atual.	Alta	RF03
Identificador	Descrição	Prioridade	Depende de
RF 06	O sistema deve gerar relatórios gerenciais sobre o fluxo de atendimento (ex: pico de demanda, tempo médio de espera).	Média	RF03
Identificador	Descrição	Prioridade	Depende de
RF 07	O sistema deve enviar notificações ao paciente quando estiver entre os 3 a ser chamado	Alta	RF03
Identificador	Descrição	Prioridade	Depende de
RF 08	O sistema deve enviar notificações ao paciente quando for chamado.	Alta	
Identificador	Descrição	Prioridade	Depende de

RF 09	O sistema deve permitir o cadastro de dados pessoais pelo próprio paciente	Alta	RF02
Identificador	Descrição	Prioridade	Depende de
RF 10	O sistema deve permitir a atualização de dados pessoais pelo próprio paciente	Alta	RF09
Identificador	Descrição	Prioridade	Depende de
RF 11	O sistema deve permitir a exportação de relatórios em formatos como PDF e CSV.	Alta	RF06
Identificador	Descrição	Prioridade	Depende de
RF 12	O sistema deve permitir ao administrador configurar os níveis de acesso e permissões de usuários.	Alta	RF02
Identificador	Descrição	Prioridade	Depende de
RF 13	O sistema deve registrar logs de ações realizadas pelos usuários (login, cadastro, edição de dados etc.).	Média	RF02
Identificador	Descrição	Prioridade	Depende de
RF 14	O sistema deve permitir a redefinição da senha	Alta	RF02
Identificador	Descrição	Prioridade	Depende de
RF 15	O sistema deve permitir que o paciente entre na fila baseado no código do hospital	Alta	RF02
Identificador	Descrição	Prioridade	Depende de
RF 16	O sistema deve retornar os 6 pacientes mais recentes da fila, sendo estes: o paciente atualmente em atendimento e os 5 pacientes que foram chamados imediatamente antes dele.	Alta	RF02
Identificador	Descrição	Prioridade	Depende de
RF 17	O sistema deve permitir que um profissional de saúde registre as informações do atendimento realizado ao paciente.	Alta	

Fonte: Autoria Própria (2025)

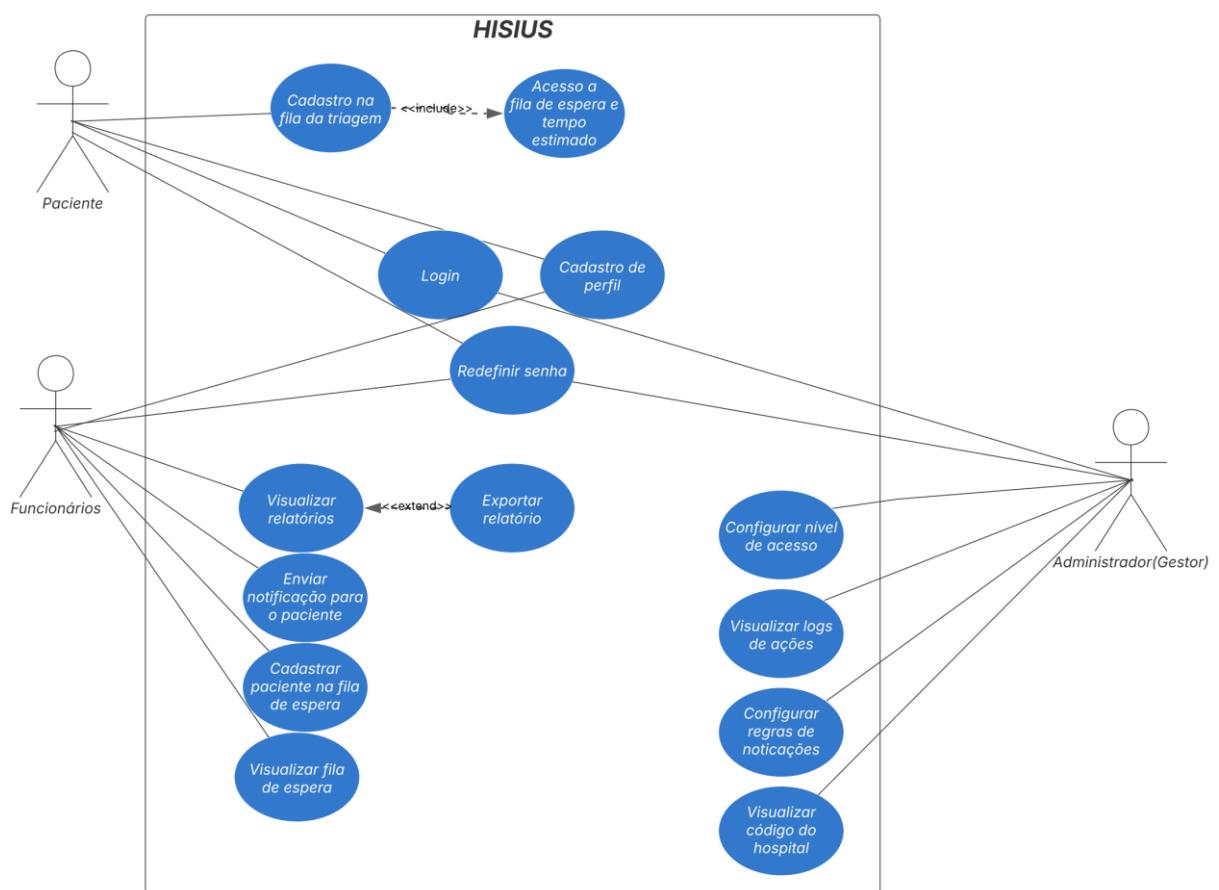
Figura B.2 – Requisitos Não Funcionais HISIUS

Identificador	Descrição	Categoria	Escopo	Prioridade	Depende de
RNF01	O sistema deve apresentar tempo de resposta inferior a 2 segundos para operações comuns (login, visualização da fila, cadastro).	Eficiência em relação ao tempo	Sistema	Alta	RF02, RF04
Identificador	Descrição	Categoria	Escopo	Prioridade	Depende de
RNF02	Senhas dos usuários devem ser armazenadas de forma criptografada utilizando algoritmos seguros (ex: bcrypt).	Segurança de Acesso	Sistema	Alta	RF01
Identificador	Descrição	Categoria	Escopo	Prioridade	Depende de
RNF03	O sistema deve ser compatível com os navegadores Google Chrome, Mozilla Firefox, Microsoft Edge e Safari, nas versões atualizadas dos últimos 12 meses.	Interoperabilidade	Sistema	Alta	
Identificador	Descrição	Categoria	Escopo	Prioridade	Depende de
RNF04	O sistema deve ser responsivo, funcionando corretamente em dispositivos móveis (Android, iOS) e desktops.	Portabilidade	Sistema	Alta	
Identificador	Descrição	Categoria	Escopo	Prioridade	Depende de
RNF05	Todas as ações críticas no sistema devem ser registradas em logs não editáveis, com acesso restrito ao administrador.	Segurança de Acesso	Funcionalidade	Alta	RF13
Identificador	Descrição	Categoria	Escopo	Prioridade	Depende de

RNF06	O sistema deve ser projetado para permitir expansão futura com o aumento no número de usuários, módulos ou dados, sem a necessidade de reestruturação completa.	Manutenibilidade	Sistema	Alta	
Identificador	Descrição	Categoria	Escopo	Prioridade	Depende de
RNF07	A interface deve ter design atrativo, com cores, ícones e elementos visuais que favoreçam a experiência do usuário.	Atratividade	Sistema	Alta	

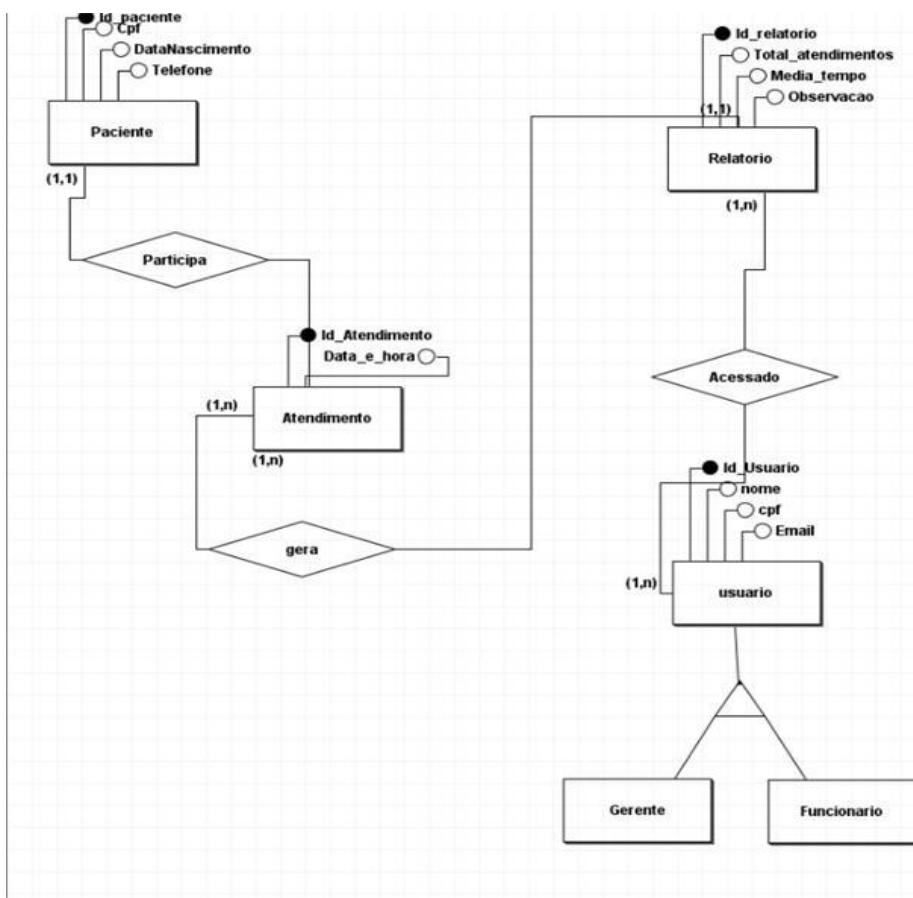
Fonte: Autoria Própria (2025)

Figura B.3 – Diagrama de Caso de Uso



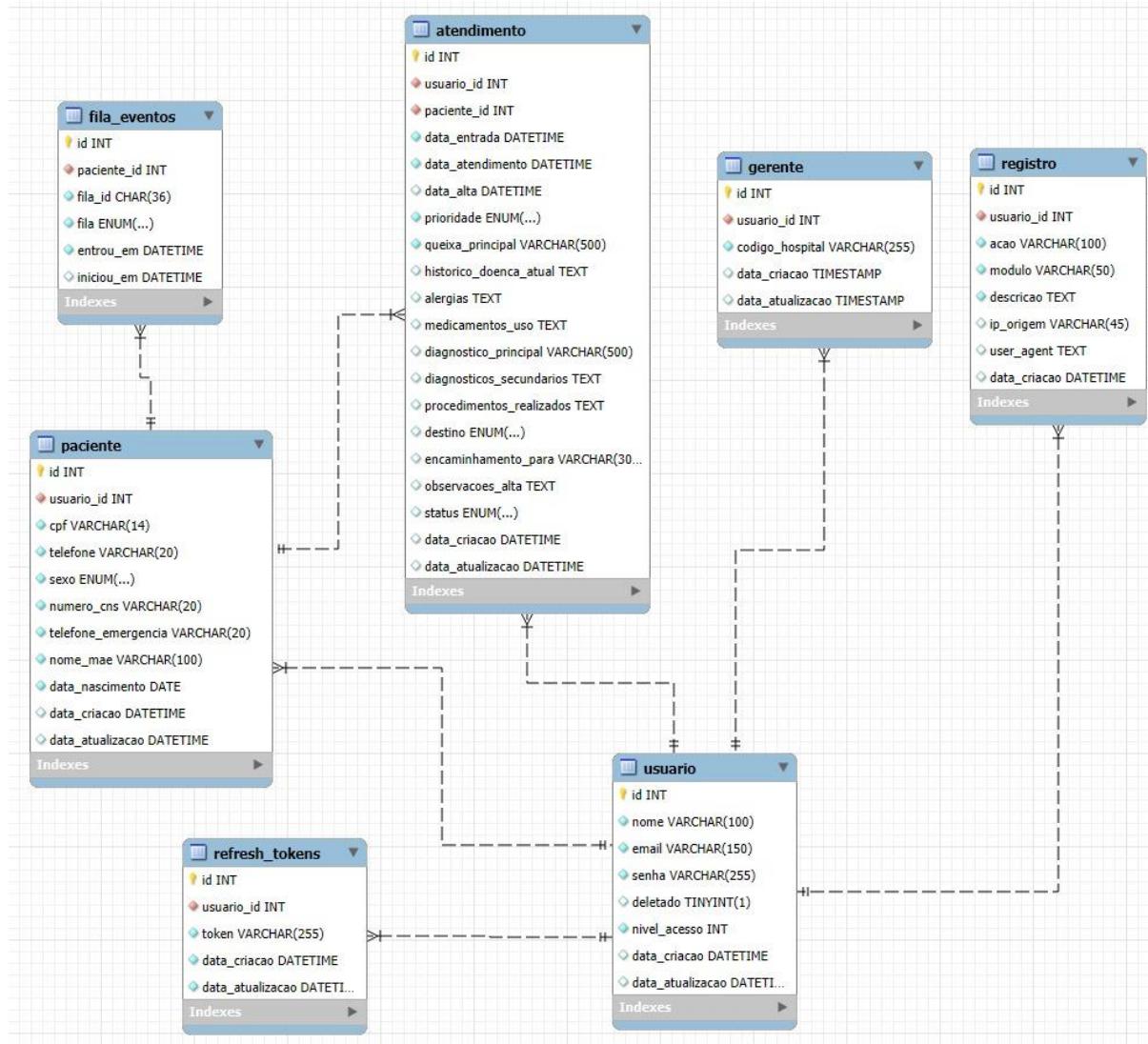
Fonte: Autoria Própria (2025)

Figura B.4 – DER (Diagrama Entidade-Relacionamento)



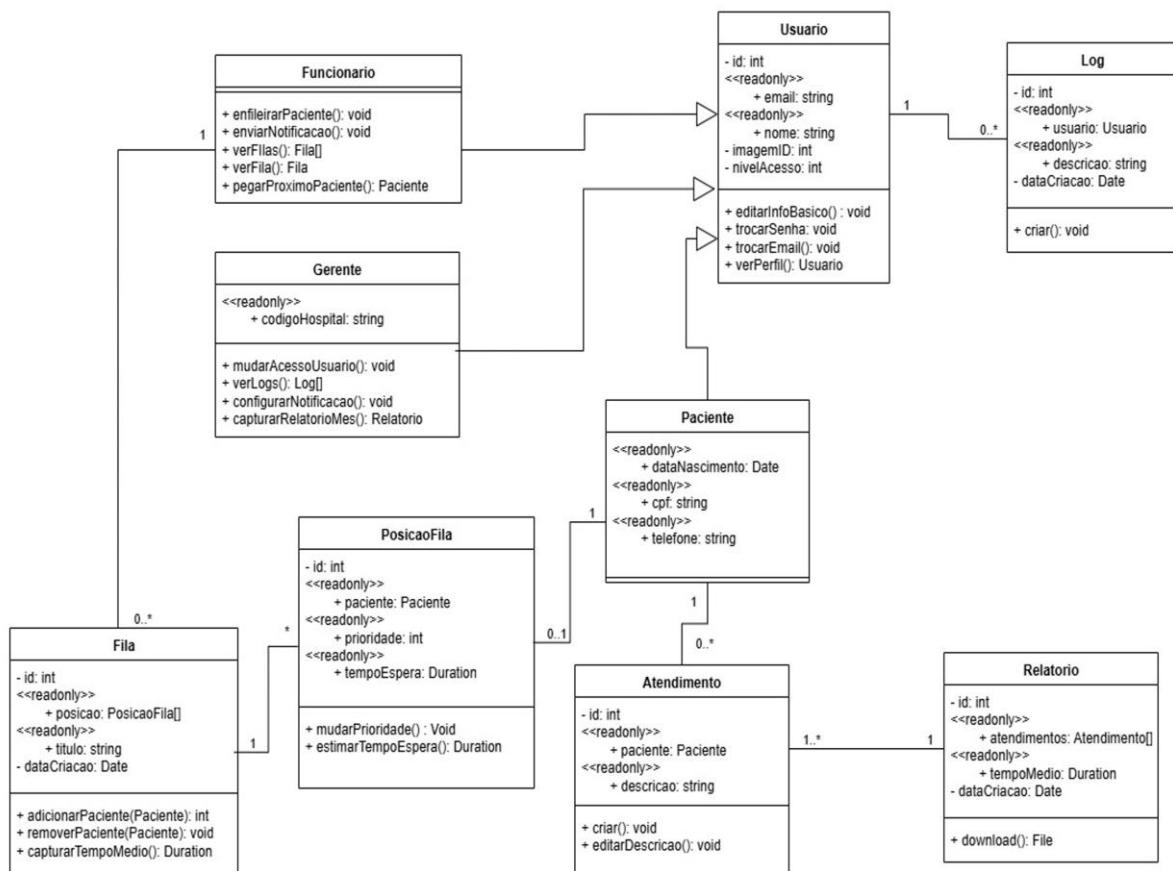
Fonte: Autoria Própria (2025)

Figura B.5 – MER (Modelo Entidade-Relacionamento)



Fonte: Autoria Própria (2025)

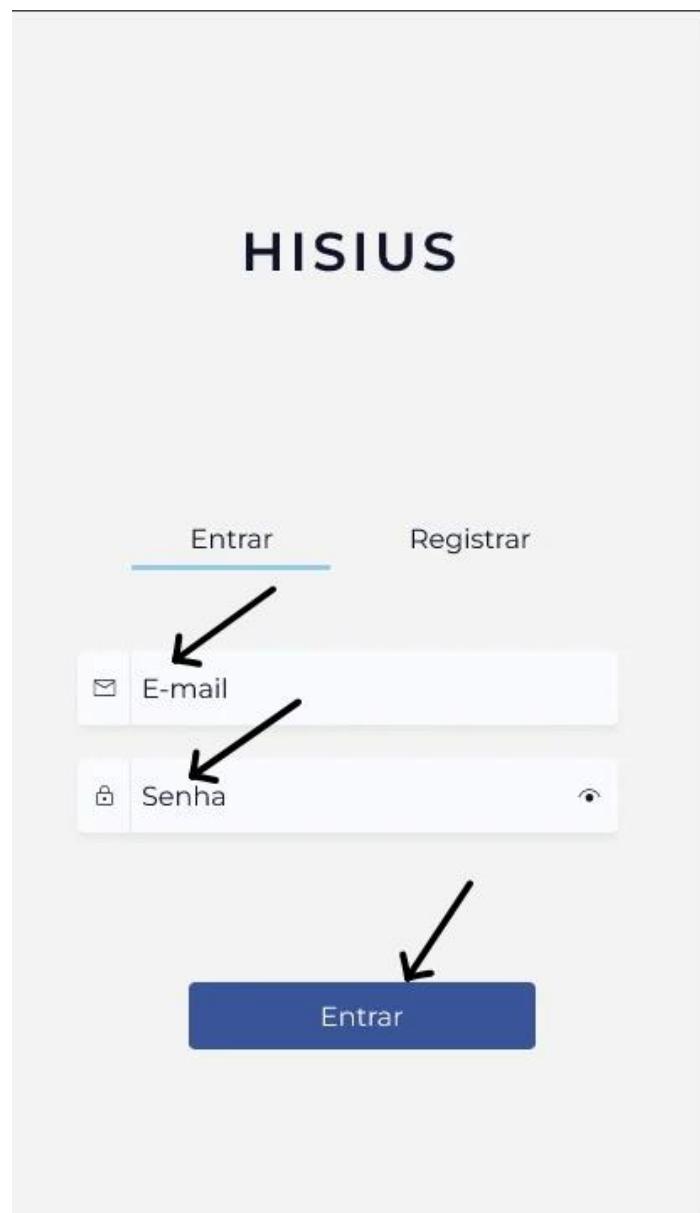
Figura B.6 – Diagrama de Classes



Fonte: Autoria Própria (2025)

ANEXO C – Telas Pacientes

Figura C.1 – Realizar Login no App



Fonte: Autoria Própria (2025)

A figura 8 ilustra o comportamento da aplicação durante o processo de Realizar Login no App. O processo se inicia ao abrir o App. Em seguida, será necessário colocar os dados solicitados. Ao clicar em "Entrar", o usuário é redirecionado para a tela inicial do app.

Figura C.2 – Entrar com Código do Hospital



Fonte: Autoria Própria (2025)

A figura 9 ilustra o comportamento da aplicação durante o processo de Entrar com o Código do Hospital. O processo se inicia após realizar o login. Em seguida, será necessário colocar o código do hospital. Ao clicar em "Entrar", o usuário é redirecionado para a tela inicial do app.

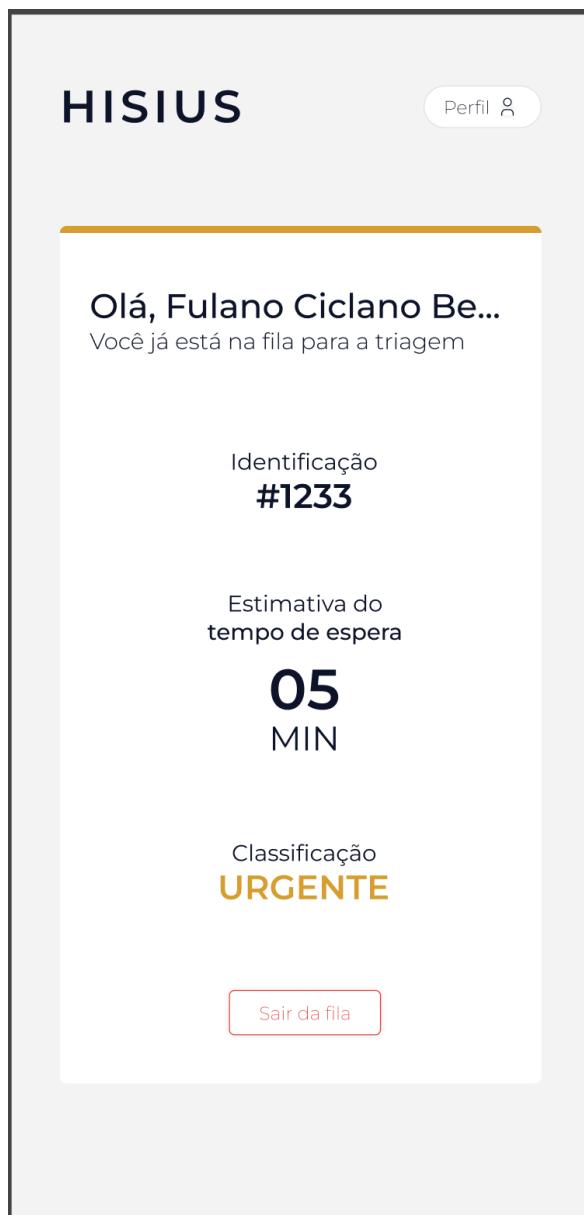
Figura C.3 – Fila de Espera da Triagem



Fonte: Autoria Própria (2025)

A figura 10 ilustra o comportamento da aplicação durante o processo de Fila de Espera da Triagem. O processo se inicia após adicionar o código do hospital. Em seguida, será mostrado o tempo estimado de esperar para passar na triagem. Ao clicar em "Sair da Fila", o usuário será retirado da fila de espera.

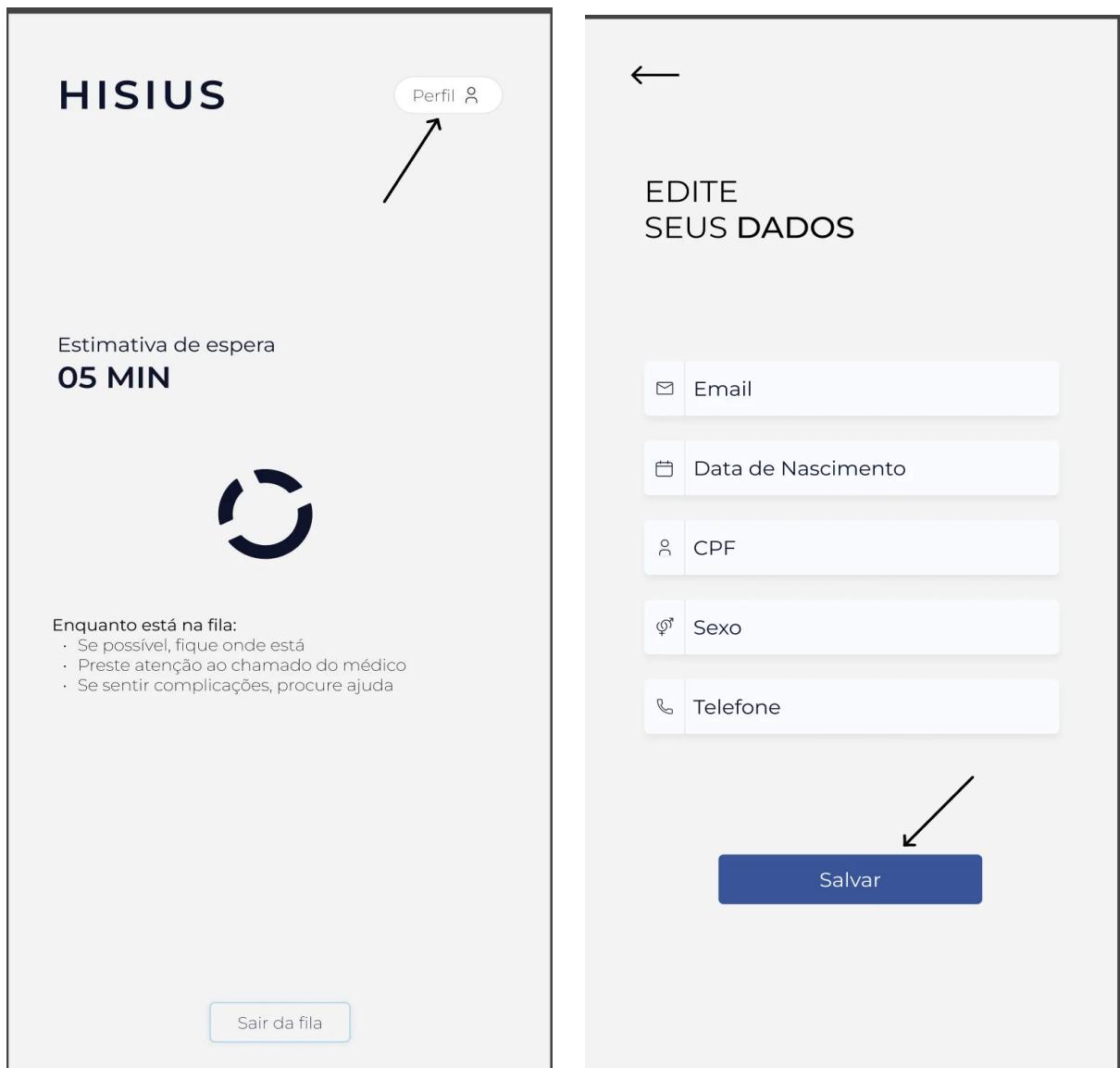
Figura C.4 – Fila Para Atendimento Médico



Fonte: Autoria Própria (2025)

A figura 11 ilustra o comportamento da aplicação durante o processo de Fila Para Atendimento Médico. O processo se inicia após passar na triagem. Em seguida, será mostrado o tempo estimado de esperar para passar no atendimento com o médico, sua classificação de risco e a identificação. Ao clicar em "Sair da Fila", o usuário será retirado da fila de espera do atendimento.

Figura C.5 – Editar Dados Pessoais

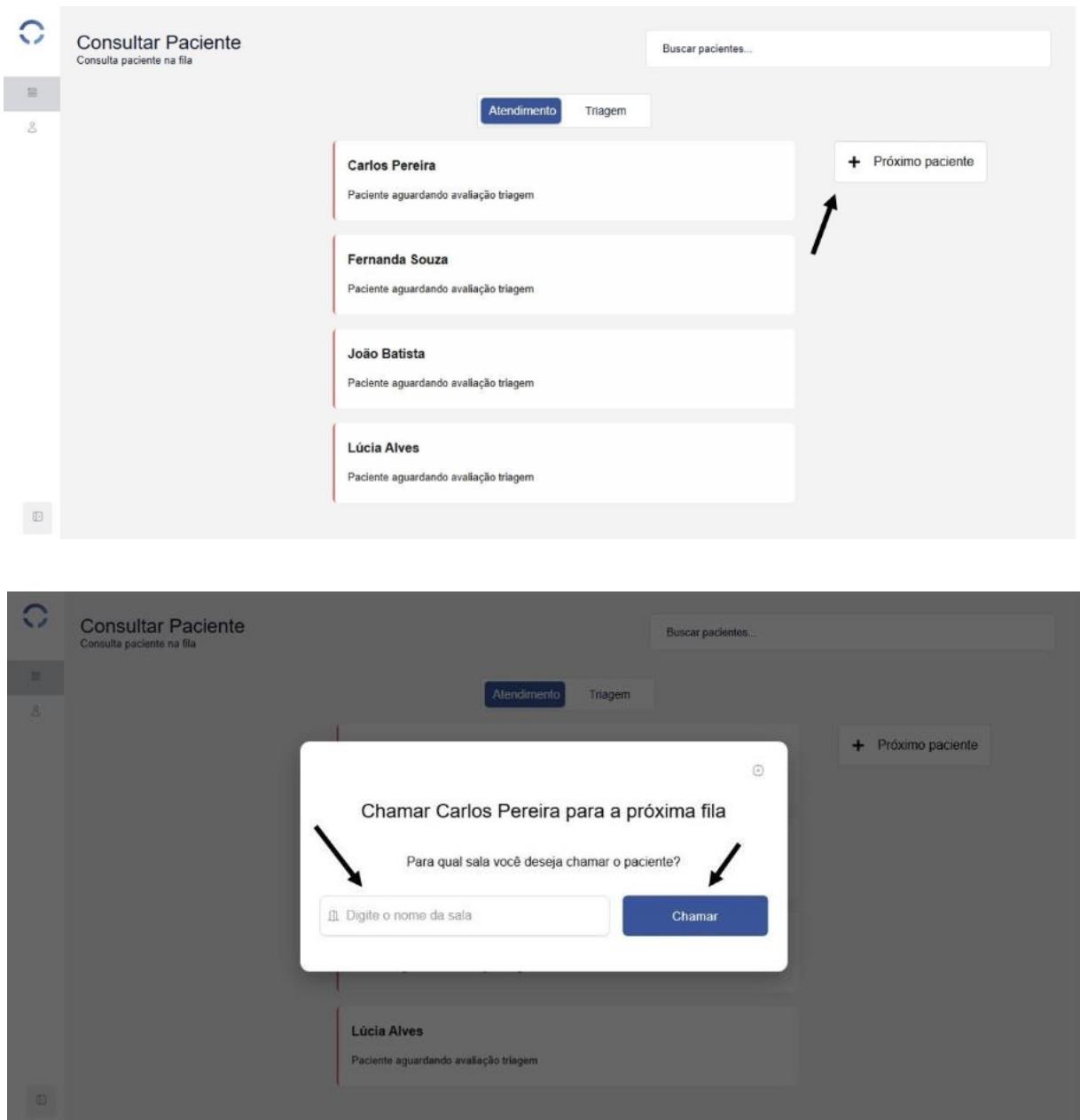


Fonte: Autoria Própria (2025)

A figura 12 ilustra o comportamento da aplicação durante o processo de edição de dados pessoais. O processo se inicia após adicionar o código do hospital. Em seguida, será necessário colocar os dados para a mudança. Ao clicar em "Salvar", será salvo os dados que o usuário adicionou logo acima.

ANEXO D – Telas Gestores/Funcionários

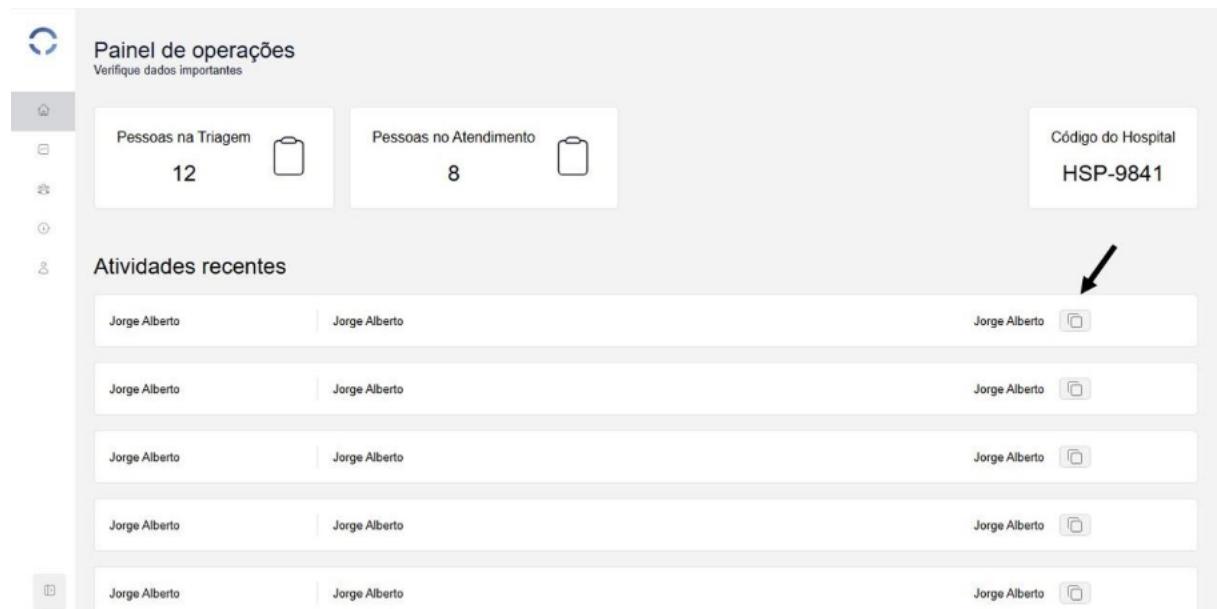
Figura D.1 – Adicionar a Sala que o Paciente Será Atendido



Fonte: Autoria Própria (2025)

A figura C.1 ilustra o comportamento da aplicação durante o processo de adicionar sala que o paciente será atendido. O processo se inicia após clicarmos em próximo paciente. Em seguida, será necessário colocar a sala que o paciente será atendido. Ao clicar em "Chamar", o paciente será chamado para ir à sala designada.

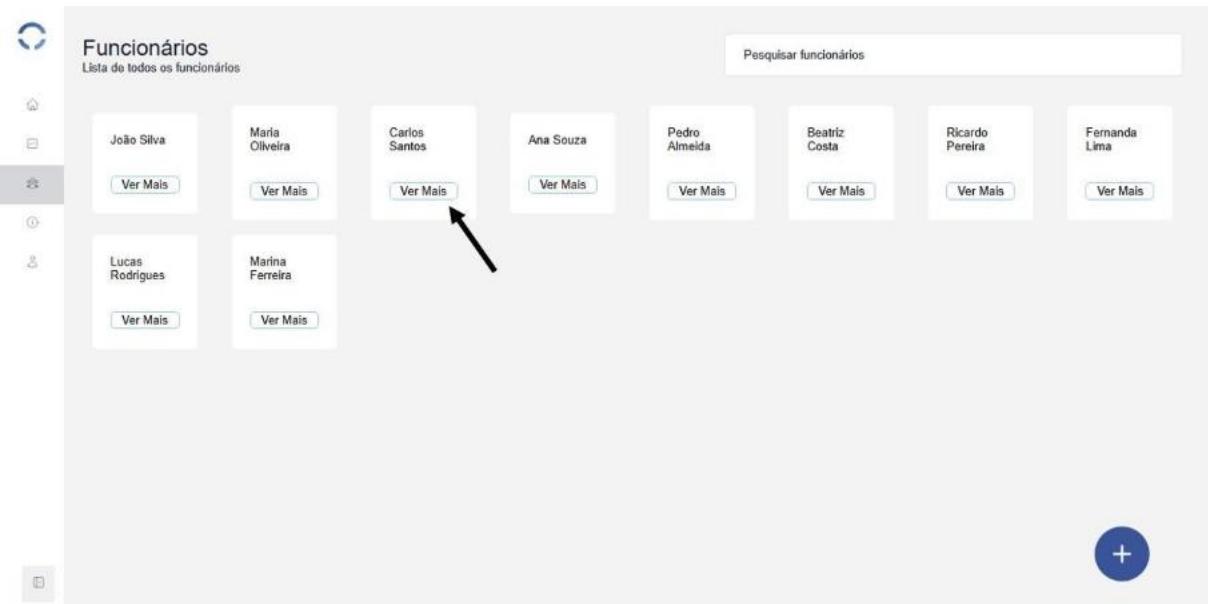
Figura D.2 - Copiar Dados do Paciente

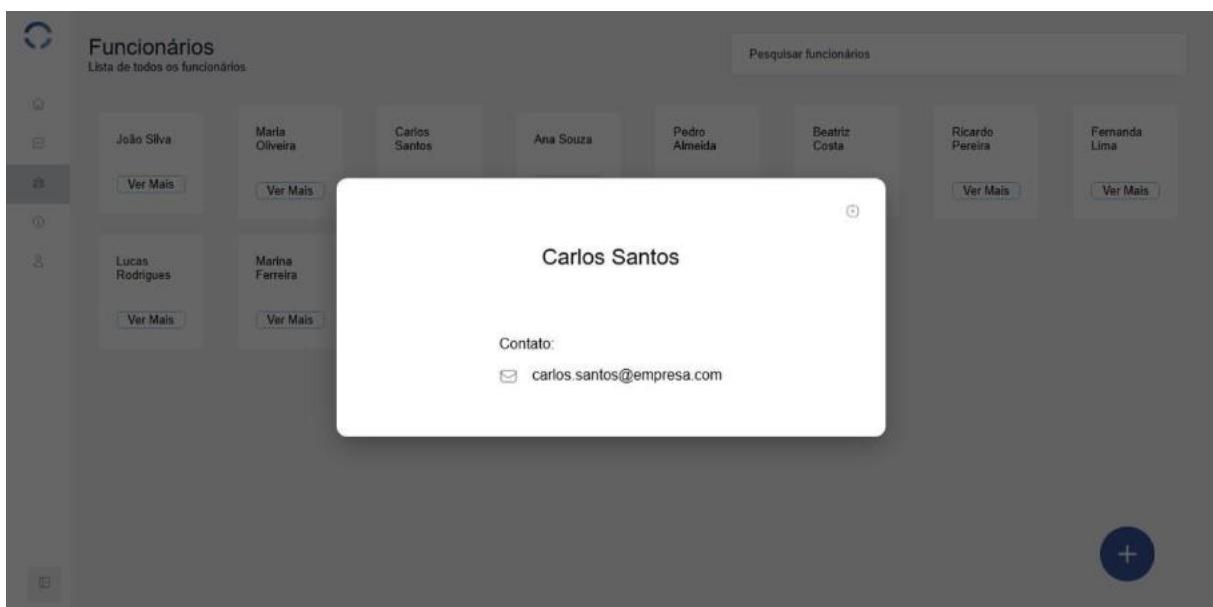


Fonte: Autoria Própria (2025)

A figura C.2 ilustra o comportamento da aplicação durante o processo de copiar dados do paciente. O processo se inicia após clicarmos no botão de copiar. Em seguida, será salvo na área de transferência os dados do paciente.

Figura D.3 – Ver informações do Funcionário

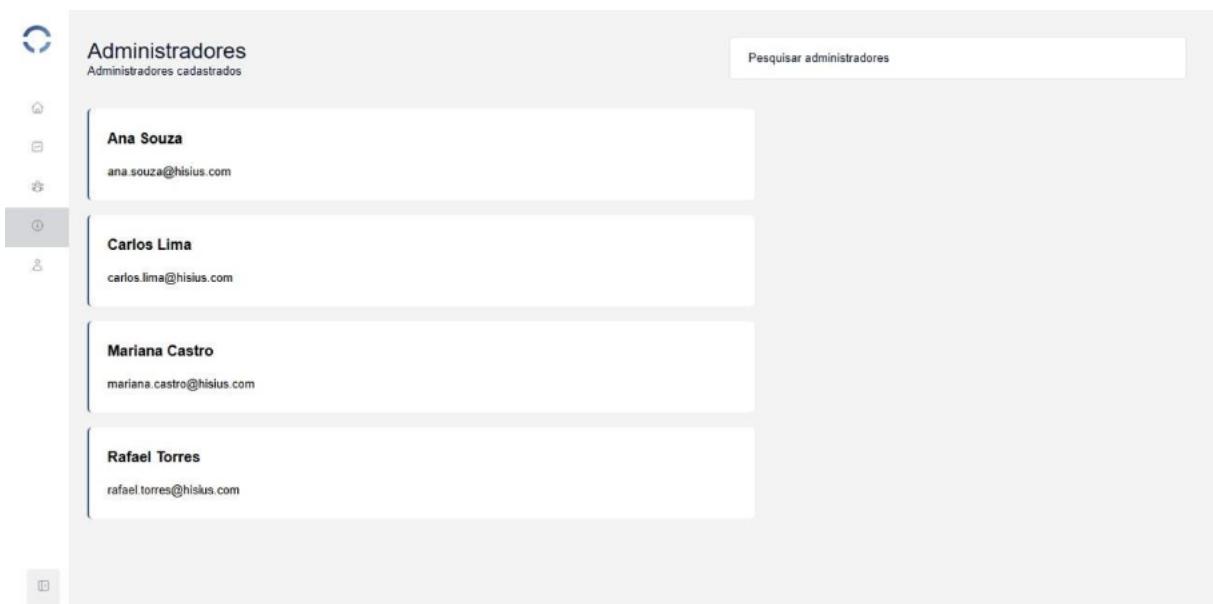




Fonte: Autoria Própria (2025)

A figura C.3 ilustra o comportamento da aplicação durante o processo de ver informações do funcionário. O processo se inicia após clicarmos no botão de "ver mais". Em seguida, será mostrado o nome completo do funcionário e o contato salvo do próprio.

Figura D.4 - Ver Administradores



Fonte: Autoria Própria (2025)

A figura C.4 ilustra o comportamento da aplicação durante o processo de ver os administradores. O processo se inicia após clicarmos na aba administradores. Em seguida, será mostrado todos os administradores do hospital.

Figura D.5 – Ver Relatório



Fonte: Autoria Própria (2025)

A figura C.5 ilustra o comportamento da aplicação durante o processo de ver os relatórios. O processo se inicia após clicarmos na aba relatórios. Em seguida, será mostrado todos as informações do relatório, podendo ver semana, mensal e diário.

Figura D.6 – Adicionar Funcionário na Equipe

Fonte: Autoria Própria (2025)

A figura C.6 ilustra o comportamento da aplicação durante o processo de adicionar funcionários a equipe. O processo se inicia após clicarmos no botão "+". Em seguida, será mostrado o token, tendo o botão de copiar para copiarmos e enviarmos o token para o funcionário.

ANEXO E – Códigos do Sistema

routes.ts

```

import express from "express";
import cors from "cors";
import morgan from "morgan";

import MaintenanceMiddleware from "./middlewares/Maintenance";
import "express-async-errors";
import cookieParser from "cookie-parser";

import { ErrorMiddleware } from "./middlewares/Error";
import { NotFoundMiddleware } from "./middlewares/NotFound";
import UserRoute from "./routes/UserRoute";
import ManagerRoute from "./routes/ManangerRoute";
import EnvRoute from "./routes/EnviromentRoute";
import AuthRoute from "./routes/AuthRoute";
import QueueRoute from "./routes/QueueRoute";
import PatientRoute from "./routes/PatientRoute";
import ReportRoute from "./routes/ReportRoute";
import EmployeeRoute from "./routes/EmployeeRoute";
import AttendanceRoute from "./routes/AttendanceRoute";
import LogRoute from "./routes/LogRoute";

import { setupSwagger } from "../swagger";

const app = express();

//ROTAS PARA PERMITIDAS
const allowedOrigins = [
  "http://localhost:5173",
  "http://localhost:5174",
  "http://localhost:8088",
  "http://localhost:8081",
];

/*
  MIDDLEWARES
*/
app.use(
  cors({
    origin: function (origin, callback) {
      if (!origin || allowedOrigins.includes(origin)) {
        callback(null, true);
      } else {
        callback(null, false);
      }
    }
  })
);

```

```

        callback(new Error("Not allowed by CORS"));
    }
},
credentials: true,
})
);
;

app.use(cookieParser());
app.set("trust proxy", true);
app.use(express.json());

app.use(morgan("dev"));
app.use(MaintenanceMiddleware);

/*
  ROTAS
*/
app.use(EnvRoute);
app.use("/queue", QueueRoute);
app.use("/reports", ReportRoute);
app.use("/auth", AuthRoute);
app.use("/users", UserRoute);
app.use("/admins", ManagerRoute);
app.use("/patients", PatientRoute);
app.use("/attendances", AttendanceRoute);
app.use("/employees", EmployeeRoute);
app.use("/logs", LogRoute);
setupSwagger(app);

/*
  ERROS E 404
*/
app.use(ErrorMiddleware);
app.use(NotFoundMiddleware); // Sempre o último

export default app;

```

server.ts

```

import "reflect-metadata";
import http from "http";
import logger from "./config/Logger";
import { ApiEnviroment } from "./enums/Api/ApiEnviroment";
import app from "./routes";

```

```

import { generateASCII } from "./utils/NameGenerator";
import packageJson from "../package.json";
import { initDB, disconnectDB } from "./database/Connection";
import { initializeModels } from "./database/models/index";
import { initSMTP } from "./config/Ssmtp";
import { connectRedis } from "./config/Redis";
import { SocketServer } from "./config/SocketServer";

const API_NAME = packageJson.name;
const APP_VERSION = packageJson.version;

let server: http.Server | null = null;

/**
 * Finaliza aplicação de forma segura
 */
const stopServer = async (isError = false) => {
  try {
    logger.warn("Parando a aplicação...");

    if (server) {
      logger.warn("Fechando servidor HTTP...");
      server.close();
    }

    logger.warn("Fechando conexão com banco de dados...");
    await disconnectDB();

    logger.warn("Aplicação encerrada!");
    process.exit(isError ? 1 : 0);
  } catch (err) {
    logger.error("Erro ao parar aplicação:", err);
    process.exit(1);
  }
};

function getHorizontalSize(asciiArt: string): number {
  return Math.max(...asciiArt.split("\n").map((line) => line.length));
}

/**
 * Inicializa servidor:
 * - Gera ASCII
 * - Conecta no banco

```

```

* - Exibe infos
* - Sobe o servidor Express
*/
const startServer = async () => {
  try {
    logger.info("Capturando variáveis de ambiente...");
    const { CONFIG } = await import("./config/Env");

    const API_VERSION = "v" + CONFIG.apiVersion;
    const ENVIRONMENT =
      CONFIG.environment === "dev" ? ApiEnviroment.DEV : ApiEnviroment.PROD;
    const PORT = CONFIG.apiPort;

    logger.info("Conectando com o banco de dados...");
    const sequelize = await initDB(CONFIG);
    logger.info("Inicializando modelos...");
    await initializeModels(sequelize);
    logger.info("Conectando com serviço SMTP...");
    await initSMTP(CONFIG);
    logger.info("Conectando com Redis...");
    await connectRedis();

    const asciiArt = await generateASCII(API_NAME);
    const horizontalSize = getHorizontalSize(asciiArt) + 1;

    console.log(asciiArt);
    console.log("=".repeat(horizontalSize));
    console.log(`Versão do app: ${APP_VERSION}`);
    console.log(`Versão API: ${API_VERSION}`);
    console.log(`Ambiente: ${ENVIRONMENT}`);
    console.log("=".repeat(horizontalSize));

    server = app.listen(PORT, async () => {
      logger.info(`Aplicação iniciada na porta ${PORT}`);
      if (ENVIRONMENT === ApiEnviroment.DEV) {
        logger.info(`API rodando em: http://localhost:${PORT}`);
      }

      logger.info("Conectando o Websocket...");
      await SocketServer.init(server);
    });

    process.on("SIGINT", () => stopServer(false));
    process.on("SIGTERM", () => stopServer(false));
  }
}

```

```

} catch (err: any) {
  logger.error("Não foi possível iniciar a aplicação:");
  logger.error(err instanceof Error ? err.message : err);
  logger.error(err instanceof Error ? err.stack : "");
  await stopServer(true);
}
};

startServer();

```

index.d.ts

```

import { Request } from "express";
import { UserRole } from "../../enums/User/UserRole";

declare global {
  namespace Express {
    interface Request {
      user?: { id: number; role?: UserRole; email?: string };
    }
  }
}

```

Env.ts

```

import * as dotenv from "dotenv";
dotenv.config();

const requiredVars = [
  "ENVIRONMENT",
  "API_VERSION",
  "API_PORT",
  "MAINTENANCE",
  "MYSQL_HOST",
  "MYSQL_PORT",
  "MYSQL_USER",
  "MYSQL_PASSWORD",
  "MYSQL_DATABASE",
  "ACCESS_TOKEN_SECRET",
  "REFRESH_TOKEN_SECRET",
  "RESET_PASS_TOKEN_SECRET",
  "RESET_EMAIL_TOKEN_SECRET",
  "GENERATE_TOKEN_SECRET",
]

```

```

"SMTP_HOST",
"SMTP_USER",
"SMTP_PASS",
"SMTP_SECURE",
"SMTP_PORT",
"SMTP_CHARSET",
"EMAIL_FROM",
"REDIS_URL",
"FRONTEND_URL",
"RESET_TOKEN_EXPIRES_MIN",
] as const;

type RequiredEnv = {
  [K in (typeof requiredVars)[number]]: string;
};

function loadEnv(): RequiredEnv {
  const missing: string[] = [];
  const env: Partial<RequiredEnv> = {};

  for (const key of requiredVars) {
    const value = process.env[key];
    if (value === undefined) {
      missing.push(key);
      continue;
    }
    env[key] = value;
  }

  if (missing.length > 0) {
    throw new Error("Variáveis de ambiente ausentes: " + missing.join(", "));
  }

  return env as RequiredEnv;
}

export const ENV = loadEnv();

export const CONFIG = {
  environment: ENV.ENVIRONMENT,
  apiVersion: Number(ENV.API_VERSION),
  apiPort: Number(ENV.API_PORT),
  maintenance: ENV.MAINTENANCE === "true",
  mysql: {

```

```

host: ENV.MYSQL_HOST,
port: Number(ENV.MYSQL_PORT),
user: ENV.MYSQL_USER,
password: ENV.MYSQL_PASSWORD,
database: ENV.MYSQL_DATABASE,
},
tokens: {
  accessSecret: ENV.ACCESS_TOKEN_SECRET,
  refreshSecret: ENV.REFRESH_TOKEN_SECRET,
  resetPassSecret: ENV.RESET_PASS_TOKEN_SECRET,
  resetEmailSecret: ENV.RESET_EMAIL_TOKEN_SECRET,
  generateSecret: ENV.GENERATE_TOKEN_SECRET,
},
redis: {
  url: ENV.REDIS_URL,
},
smtp: {
  host: ENV.SMTP_HOST,
  user: ENV.SMTP_USER,
  pass: ENV.SMTP_PASS,
  secure: ENV.SMTP_SECURE === "true",
  port: Number(ENV.SMTP_PORT),
  charset: ENV.SMTP_CHARSET,
  from: ENV.EMAIL_FROM,
},
frontendUrl: ENV.FRONTEND_URL,
resetTokenExpiresMin: Number(ENV.RESET_TOKEN_EXPIRES_MIN),
};

```

Logger.ts

```

import winston from "winston";
import path from "path";
import fs from "fs";

const logsDirectory = path.join(__dirname, "../../logs");
if (!fs.existsSync(logsDirectory)) {
  fs.mkdirSync(logsDirectory);
}

const logPath = path.join(logsDirectory, "app.log");

const addEmote = (level: string): string => {
  switch (level) {

```

```

case "info":
  return "ℹ️";
case "warn":
  return "⚠️";
case "error":
  return "✖️";
case "debug":
  return "🔍";
default:
  return "📝";
}

};

const Logger = winston.createLogger({
  level: "debug",
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.printf(({ timestamp, level, message }) => {
      const emote = addEmote(level);
      const coloredLevel = winston.format.colorize().colorize(level, level);
      return `${timestamp} ${emote} ${coloredLevel} : ${message}`;
    })
  ),
  transports: [
    new winston.transports.Console({
      format: winston.format.combine(
        winston.format.timestamp(),
        winston.format.printf(({ timestamp, level, message }) => {
          const emote = addEmote(level);
          const coloredLevel = winston.format.colorize().colorize(level, level);
          return `${timestamp} ${emote} ${coloredLevel} : ${message}`;
        })
      ),
    }),
    new winston.transports.File({
      filename: logPath,
      format: winston.format.combine(
        winston.format.timestamp(),
        winston.format.printf(({ timestamp, level, message }) => {
          const emote = addEmote(level);
          return `${timestamp} ${emote} ${level} : ${message}`;
        })
      ),
    })
  ]
});

```

```
  }),
],
});
```

```
export default Logger;
```

Redis.ts

```
import Redis from "ioredis";
import { CONFIG } from "./Env";
import Logger from "./Logger";

let redis: Redis | null = null;

export const connectRedis = async (): Promise<Redis> => {
  redis = new Redis(CONFIG.redis.url, {
    retryStrategy(times) {
      if (times > 10) {
        Logger.error(
          "Não foi possível conectar ao Redis após várias tentativas"
        );
        return null;
      }
      return Math.min(times * 100, 3000);
    },
  });
}

redis.on("connect", () => Logger.info("Redis (ioredis) conectado"));
redis.on("error", (err) => Logger.error("Redis error:", err));
redis.on("reconnecting", () => Logger.info("Redis reconectando..."));

try {
  await redis.ping();
} catch (err) {
  Logger.error("Falha ao pingar o Redis:", err);
  throw err;
}

return redis;
};

export const getRedis = (): Redis => {
  if (!redis) {
    throw new Error("Redis não conectado. Chame connectRedis() primeiro.");
  }
}
```

```

    }
    return redis;
};


```

Smtp.ts

```

import nodemailer, { Transporter } from "nodemailer";
import Logger from "../config/Logger";

let transporter: Transporter | null = null;
let fromAddress: string;

export const initSMTP = async (CONFIG: any): Promise<Transporter> => {
  if (transporter) return transporter;

  const { host, port, user, pass, secure, from } = CONFIG.smtp;

  transporter = nodemailer.createTransport({
    host,
    port,
    secure,
    auth: { user, pass },
    requireTLS: true
  });

  try {
    await transporter.verify();
    Logger.info("Conexão com SMTP realizada com sucesso!");
    fromAddress = from;
  } catch (error) {
    Logger.error("Erro ao conectar ao SMTP:", error);
    throw error;
  }

  return transporter;
};

export const getTransporter = (): {
  transporter: Transporter;
  from: string;
} => {
  if (!transporter || !fromAddress)
    throw new Error("SMTP não configurado. Chame initSMTP primeiro.");
  return { transporter, from: fromAddress };
}

```

```
};
```

SocketServer.ts

```
import { Server } from "socket.io";
import Logger from "./Logger";
import { GatewayMiddleware } from "../middlewares/GatewayMiddleware";
import { UserRole } from "../enums/User/UserRole";

export class SocketServer {
    private static io: Server;

    static init(httpServer: any) {
        this.io = new Server(httpServer, { cors: { origin: "*" } });
        this.io.use(GatewayMiddleware);

        this.io.on("connection", (socket) => {
            const user = socket.data.user;
            if (!user) return socket.disconnect();

            socket.join(`user:${user.id}`);
            if (user.role === UserRole.ADMIN) socket.join("admin:panel");

            Logger.info(`Usuário ${user.id} (${user.role}) conectado`);

            socket.on("disconnect", () => {
                Logger.info(`Usuário ${user.id} desconectou`);
            });
        });

        Logger.info("WebSocket inicializado");
    }

    static getIo(): Server {
        if (!this.io) throw new Error("WebSocket não inicializado");
        return this.io;
    }
}
```

AttendanceController.ts

```
import { plainToInstance } from "class-transformer";
import { NextFunction, Request, Response } from "express";
```

```

import { SuccessResponse } from "../utils/responses/SuccessResponse";
import { AttendanceService } from "../service/AttendanceService";
import { AttendanceQueryDto } from "../dtos/attendance/AttendanceQueryDto";
import { AttendanceParamDto } from "../dtos/attendance/AttendanceParamDto";
import { AttendanceDto } from "../dtos/attendance/AttendanceDto";

const attendanceService = new AttendanceService();

export class AttendanceController {
  static async getAttendances(req: Request, res: Response, next: NextFunction) {
    try {
      const queryDto = plainToInstance(AttendanceQueryDto, req.query);
      const paramDto = plainToInstance(AttendanceParamDto, req.params);

      const attendances = await attendanceService.getPaginated({
        page: queryDto.page,
        limit: queryDto.limit,
        patientId: paramDto.patientId,
      });
      res
        .status(200)
        .json(
          SuccessResponse(
            attendances,
            "Registro de atendimentos capturados com sucesso",
            200
          )
        );
    } catch (err) {
      next(err);
    }
  }

  static async getAttendance(req: Request, res: Response, next: NextFunction) {
    try {
      const paramDto = plainToInstance(AttendanceParamDto, req.params);

      const attendance = await attendanceService.getById(paramDto.attendanceId);
      res
        .status(200)
        .json(
          SuccessResponse(
            attendance,
            "Registro de atendimento capturado com sucesso",
            200
          )
        );
    } catch (err) {
      next(err);
    }
  }
}

```

```

    200
  )
);
} catch (err) {
  next(err);
}
}

static async update(req: Request, res: Response, next: NextFunction) {
try {
  const dto = plainToInstance(AttendanceDto, req.body);
  const paramDto = plainToInstance(AttendanceParamDto, req.params);

  await attendanceService.update(dto, paramDto.attendanceId);
  res
    .status(200)
    .json(SuccessResponse(null, "Registro de atendimento atualizado", 200));
} catch (err) {
  next(err);
}
}

static async deleteAttendance(
  req: Request,
  res: Response,
  next: NextFunction
) {
try {
  const paramDto = plainToInstance(AttendanceParamDto, req.params);

  await attendanceService.delete(paramDto.attendanceId);
  res
    .status(200)
    .json(SuccessResponse(null, "Registro de atendimento deletado", 200));
} catch (err) {
  next(err);
}
}
}

```

AuthController.ts

```

import { plainToInstance } from "class-transformer";
import { UserDTO } from "../dtos/user/UserDto";

```

```

import { NextFunction, Request, Response } from "express";
import { SuccessResponse } from "../utils/responses/SuccessResponse";
import { AuthService } from "../service/AuthService";
import { BadRequestError } from "../utils/errors/BadRequestError";

const authService = new AuthService();

export class AuthController {
  static async login(req: Request, res: Response, next: NextFunction) {
    try {
      const dto = plainToInstance(UserDTO, req.body);
      const response = await authService.login(dto.email, dto.password);
      const payload = { ...response.user, accessToken: response.accessToken };
      AuthController.createRefreshCookie(res, response.refreshToken);

      return res
        .status(200)
        .json(SuccessResponse(payload, "Logado com sucesso", 200));
    } catch (err) {
      next(err);
    }
  }

  static async logout(req: Request, res: Response, next: NextFunction) {
    try {
      const token = req.cookies?.refreshToken;
      if (!token) throw new BadRequestError("Token ausente");

      await authService.logout(token);

      res.clearCookie("refreshToken", {
        httpOnly: true,
        secure: process.env.ENVIRONMENT === "production",
        sameSite: "strict",
        path: "/auth",
      });
    }

    return res
      .status(200)
      .json(SuccessResponse(null, "Desconectado com sucesso", 200));
  } catch (err) {
    next(err);
  }
}

```

```

static async refresh(req: Request, res: Response, next: NextFunction) {
  try {
    const token = req.cookies?.refreshToken;
    if (!token) throw new BadRequestError("Token ausente");

    const { accessToken, refreshToken } = await authService.refreshToken(
      token
    );

    AuthController.createRefreshCookie(res, refreshToken);

    return res
      .status(200)
      .json(SuccessResponse({ accessToken }, "Token atualizado", 200));
  } catch (err) {
    next(err);
  }
}

static async requestEmail(req: Request, res: Response, next: NextFunction) {
  try {
    const loggedInUser = req.user;
    if (!loggedInUser) throw new BadRequestError("Acesso negado");

    const dto = plainToInstance(UserDTO, req.body);
    await authService.requestResetEmailToken(loggedInUser.id, dto.email);

    return res
      .status(200)
      .json(SuccessResponse(null, "Email enviado com sucesso.", 200));
  } catch (err) {
    next(err);
  }
}

static async changeEmail(req: Request, res: Response, next: NextFunction) {
  try {
    const loggedInUser = req.user;
    if (!loggedInUser) throw new BadRequestError("Acesso negado");

    await authService.changeEmail(loggedInUser.id, loggedInUser.email!);

    return res
  }
}

```

```
.status(200)
.json(SuccessResponse(null, "Email alterado com sucesso", 200));
} catch (err) {
next(err);
}
}

static async requestResetPassword(
req: Request,
res: Response,
next: NextFunction
) {
try {
const dto = plainToInstance(UserDTO, req.body);
await authService.requestResetPassToken(dto.email);

return res
.status(200)
.json(SuccessResponse(null, "Email enviado com sucesso.", 200));
} catch (err) {
next(err);
}
}

static async recoverPassword(
req: Request,
res: Response,
next: NextFunction
) {
try {
const loggedInUser = req.user;
if (!loggedInUser) throw new BadRequestError("Acesso negado");

const dto = plainToInstance(UserDTO, req.body);
await authService.recoverPassword(loggedInUser.id, dto.password);

return res
.status(200)
.json(SuccessResponse(null, "Senha alterada com sucesso", 200));
} catch (err) {
next(err);
}
}
```

```

static createRefreshCookie(res: Response, refresh: string) {
  res.cookie("refreshToken", refresh, {
    httpOnly: true,
    secure: process.env.ENVIRONMENT === "production",
    sameSite: "strict",
    path: "/auth",
    maxAge: 7 * 24 * 60 * 60 * 1000, //7 dias
  });
}
}
}

```

EmployeeController.ts

```

import { plainToInstance } from "class-transformer";
import { UserDTO } from "../dtos/user/UserDto";
import { NextFunction, Request, Response } from "express";
import { SuccessResponse } from "../utils/responses/SuccessResponse";
import { EmployeeService } from "../service/EmployeeService";
import { EmployeeDto } from "../dtos/employee/EmployeeDto";
import { IUserQueryParams } from "../interfaces/user/IUserQueryParams";

const employeeService = new EmployeeService();

export class EmployeeController {
  static async register(req: Request, res: Response, next: NextFunction) {
    try {
      const dto = plainToInstance(UserDTO, req.body);
      const { role, ...rest } = dto;
      const user = await employeeService.createUser(rest);

      return res
        .status(201)
        .json(SuccessResponse(user, "Usuário criado com sucesso!", 201));
    } catch (err: any) {
      next(err);
    }
  }

  static async getEmployees(req: Request, res: Response, next: NextFunction) {
    try {
      const queryParams = plainToInstance(EmployeeDto, req.query);
      const page = queryParams.page;
      const limit = queryParams.limit;
      const name = queryParams.nameFilter;
    }
  }
}

```

```
const result = await employeeService.getEmployeesPaginated({  
    page,  
    limit,  
    name,  
} as IUserQueryParams);  
  
return res  
.status(200)  
.json(  
    SuccessResponse(result, "Funcionário capturados com sucesso", 200)  
);  
}  
} catch (err: any) {  
    next(err);  
}  
}  
}  
}
```

LogController.ts

```
import { plainToInstance } from "class-transformer";
import { NextFunction, Request, Response } from "express";
import { SuccessResponse } from "../utils/responses/SuccessResponse";
import { ILogFilter } from "../interfaces/log/ILogFilter";
import { LogService } from "../service/LogService";
import { LogFilterDto } from "../dtos/log/LogFilterDto";

const logService = new LogService();

export class LogController {
    static async getLogs(req: Request, res: Response, next: NextFunction) {
        try {
            const paramDto = plainToInstance(LogFilterDto, req.query);

            const filter: ILogFilter = {
                page: paramDto.page ? Number(paramDto.page) : 0,
                limit: paramDto.limit ? Number(paramDto.limit) : 10,
                userId: paramDto.userId ? Number(paramDto.userId) : undefined,
                action: paramDto.action as string,
                module: paramDto.module as string,
            };

            const result = await logService.getLogsPaginated(filter);
            res.json(result);
        } catch (error) {
            next(error);
        }
    }
}
```

```

    return res.json(
      SuccessResponse(result, "Logs recuperados com sucesso", 200)
    );
  } catch (err) {
    next(err);
  }
}
}
}

```

ManagerController.ts

```

import { NextFunction, Request, Response } from "express";
import { ManagerService } from "../service/ManagerService";
import { UserDTO } from "../dtos/user/UserDto";
import { plainToInstance } from "class-transformer";
import { SuccessResponse } from "../utils/responses/SuccessResponse";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { ManagerDto } from "../dtos/manager/ManagerDto";
import { IUserQueryParams } from "../interfaces/user/IUserQueryParams";

const managerService = new ManagerService();

export class ManagerController {
  static async register(req: Request, res: Response, next: NextFunction) {
    try {
      const dto = plainToInstance(UserDTO, req.body);
      const { role, ...rest } = dto;
      const user = await managerService.create(rest);

      return res
        .status(201)
        .json(SuccessResponse(user, "Usuário criado com sucesso!", 201));
    } catch (err: any) {
      next(err);
    }
  }

  static async getManagers(req: Request, res: Response, next: NextFunction) {
    try {
      const loggedInUser = req.user;
      if (!loggedInUser) throw new BadRequestError("Acesso negado");

      const queryParams = plainToInstance(ManagerDto, req.query);
      const page = queryParams.page;
    }
  }
}

```

```
const limit = queryParams.limit;
const name = queryParams.nameFilter;

const result = await managerService.getAdminsPaginated(
  {
    page,
    limit,
    name,
  } as IUserQueryParams,
  loggedInUser.id
);

return res
  .status(200)
  .json(
    SuccessResponse(result, "Administradores capturados com sucesso", 200)
  );
} catch (err: any) {
  next(err);
}
}

static async generateAddEmployeToken(
  req: Request,
  res: Response,
  next: NextFunction
) {
  try {
    const loggedInUser = req.user;
    if (!loggedInUser) throw new BadRequestError("Acesso negado");

    const token: string = await managerService.generateAddEmployeToken(
      loggedInUser.id
    );
  }

  return res
    .status(201)
    .json(
      SuccessResponse(
        { token: token },
        "Informações capturadas com sucesso!",
        201
      )
    );
}
```

```

    } catch (err: any) {
      next(err);
    }
  }

  static async getHospitalCode(
    req: Request,
    res: Response,
    next: NextFunction
  ) {
    try {
      const loggedInUser = req.user;
      if (!loggedInUser) throw new BadRequestError("Acesso negado");

      const hospitalCode: string = await managerService.getHospitalCode(
        loggedInUser.id
      );
    }

    return res
      .status(201)
      .json(
        SuccessResponse(
          { hospitalCode: hospitalCode },
          "Informações capturadas com sucesso!",
          201
        )
      );
  }
}

```

PatientController.ts

```

import { plainToInstance } from "class-transformer";
import { PatientDto } from "../dtos/patient/PatientDto";
import { PatientService } from "../service/PatientService";
import { NextFunction, Request, Response } from "express";
import { SuccessResponse } from "../utils/responses/SuccessResponse";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { UserService } from "../service/UserService";

const patientService = new PatientService();

```

```
const userService = new UserService();

export class PatientController {
  static async getMyProfile(req: Request, res: Response, next: NextFunction) {
    try {
      const loggedInUser = req.user;
      if (!loggedInUser) throw new BadRequestError("Usuário não autenticado.");

      const patientProfile = await patientService.getPatientById(
        loggedInUser.id
      );
      return res.json(
        SuccessResponse(patientProfile, "Perfil do paciente encontrado", 200)
      );
    } catch (err) {
      next(err);
    }
  }

  static async updateMyProfile(
    req: Request,
    res: Response,
    next: NextFunction
  ) {
    try {
      const loggedInUser = req.user;
      if (!loggedInUser) throw new BadRequestError("Usuário não autenticado.");

      const patientDto = plainToInstance(PatientDto, req.body);
      const { name, ...dto } = patientDto;

      if (name) {
        await userService.changeName(loggedInUser.id, name);
      }

      await patientService.updatePatientById(loggedInUser.id, dto);

      return res.json(
        SuccessResponse(null, "Perfil atualizado com sucesso!", 200)
      );
    } catch (err) {
      next(err);
    }
  }
}
```

```

static async createPatient(req: Request, res: Response, next: NextFunction) {
  try {
    const loggedInUser = req.user;
    if (!loggedInUser) throw new BadRequestError("Usuário não autenticado.");

    const dto = plainToInstance(PatientDto, req.body);
    const patient = await patientService.createPatient(dto, loggedInUser.id);
    return res
      .status(201)
      .json(
        SuccessResponse(patient, "Paciente registrado com sucesso!", 201)
      );
  } catch (err) {
    next(err);
  }
}

static async getPatient(req: Request, res: Response, next: NextFunction) {
  try {
    const { id } = req.params;
    const patient = await patientService.getPatientById(Number(id));
    return res.json(SuccessResponse(patient, "Paciente encontrado.", 200));
  } catch (err) {
    next(err);
  }
}

```

QueueController.ts

```

import { NextFunction, Request, Response } from "express";
import { SuccessResponse } from "../utils/responses/SuccessResponse";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { QueueService } from "../service/QueueService";
import { QueueType } from "../enums/Queue/QueueType";
import { plainToInstance } from "class-transformer";
import { QueueDto } from "../dtos/queue/QueueDto";
import { QueueParamsDto } from "../dtos/queue/QueueParamsDto";
import { NotFoundError } from "../utils/errors/NotFoundError";
import { EnterQueueDto } from "../dtos/queue/EnterQueueDto";

const queueService = new QueueService();

```

```

export class QueueController {
  static async queueJoin(req: Request, res: Response, next: NextFunction) {
    try {
      const bodyDto = plainToInstance(EnterQueueDto, req.body);

      const loggedInUser = req.user;
      if (!loggedInUser) throw new BadRequestError("Acesso negado");

      await queueService.enqueuePatient(
        loggedInUser.id,
        QueueType.TRIAGE,
        bodyDto.hospitalCode
      );

      return res
        .status(200)
        .json(SuccessResponse(null, "Entrou na fila com sucesso", 200));
    } catch (err: any) {
      next(err);
    }
  }
}

static async getSelfInfo(req: Request, res: Response, next: NextFunction) {
  try {
    const loggedInUser = req.user;
    if (!loggedInUser) throw new BadRequestError("Acesso negado");

    const patient = await queueService.getPatientInfo(loggedInUser.id);

    return res
      .status(200)
      .json(
        SuccessResponse(patient, "Informações capturadas com sucesso", 200)
      );
  } catch (err: any) {
    next(err);
  }
}

static async getLastCalledPatients(
  req: Request,
  res: Response,
  next: NextFunction
) {

```

```

try {
    const paramDto = plainToInstance(QueueParamsDto, req.params);

    const patients = await queueService.getCalledPatients(paramDto.type);

    return res
        .status(200)
        .json(
            SuccessResponse(
                patients,
                "Pacientes chamados capturado com sucesso",
                200
            )
        );
} catch (err: any) {
    next(err);
}
}

static async getNextPatient(req: Request, res: Response, next: NextFunction) {
    try {
        const loggedInUser = req.user;
        if (!loggedInUser) throw new BadRequestError("Acesso negado");

        const paramDto = plainToInstance(QueueParamsDto, req.params);
        const dto = plainToInstance(QueueDto, req.body);

        const patient = await queueService.getNextPatient(
            paramDto.type,
            dto.room,
            loggedInUser.id
        );

        if (!patient) throw new NotFoundError("Não há nenhum paciente na fila");

        return res
            .status(200)
            .json(
                SuccessResponse(patient, "Próximo paciente chamado com sucesso", 200)
            );
    } catch (err: any) {
        next(err);
    }
}

```

```

static async moveToNextQueue(
  req: Request,
  res: Response,
  next: NextFunction
) {
  try {
    const paramDto = plainToInstance(QueueParamsDto, req.params);
    const dto = plainToInstance(QueueDto, req.body);

    await queueService.moveToTreatment(
      paramDto.patientId,
      dto.classification
    );

    return res
      .status(200)
      .json(SuccessResponse(null, "Paciente movido com sucesso", 200));
  } catch (err: any) {
    next(err);
  }
}

static async getQueueCount(req: Request, res: Response, next: NextFunction) {
  try {
    const paramDto = plainToInstance(QueueParamsDto, req.params);

    const count = await queueService.getQueueCount(paramDto.type);

    return res
      .status(200)
      .json(SuccessResponse({ count }, "Dados coletados com sucesso", 200));
  } catch (err: any) {
    next(err);
  }
}

static async finishTreatment(
  req: Request,
  res: Response,
  next: NextFunction
) {
  try {
    const paramDto = plainToInstance(QueueParamsDto, req.params);
  
```

```
await queueService.finishTreatment(paramDto.patientId);

return res
  .status(200)
  .json(SuccessResponse(null, "Atendimento finalizado com sucesso", 200));
} catch (err: any) {
  next(err);
}
}

static async queueLeave(req: Request, res: Response, next: NextFunction) {
try {
  const loggedInUser = req.user;
  if (!loggedInUser) throw new BadRequestError("Acesso negado");

  await queueService.dequeuePatient(loggedInUser.id, true);

  return res
    .status(200)
    .json(SuccessResponse(null, "Saiu da fila com sucesso", 200));
} catch (err: any) {
  next(err);
}
}

static async updateQueueClassification(
  req: Request,
  res: Response,
  next: NextFunction
) {
try {
  const paramDto = plainToInstance(QueueParamsDto, req.params);
  const dto = plainToInstance(QueueDto, req.body);

  await queueService.updateClassification(
    paramDto.patientId,
    dto.classification
  );

  return res
    .status(200)
    .json(
      SuccessResponse(
        null,
```

```
    "Classificação de risco alterada com sucesso",
    200
)
);
} catch (err: any) {
    next(err);
}
}

static async getPatientsByRoom(
    req: Request,
    res: Response,
    next: NextFunction
) {
    try {
        const queryDto = plainToInstance(QueueDto, req.query);
        const paramDto = plainToInstance(QueueParamsDto, req.params);

        const enqueuedPatients = await queueService.getPatientsInRoom(
            paramDto.type,
            queryDto.page,
            queryDto.limit,
            queryDto.classification,
            queryDto.nameFilter
        );
    }

    return res
        .status(200)
        .json(
            SuccessResponse(
                enqueuedPatients,
                "Dados das salas capturado com sucesso!",
                200
            )
        );
} catch (err: any) {
    next(err);
}
}

static async getPatientsByQueue(
    req: Request,
    res: Response,
    next: NextFunction
```

```

) {
  try {
    const queryDto = plainToInstance(QueueDto, req.query);
    const paramDto = plainToInstance(QueueParamsDto, req.params);

    const enqueuedPatients = await queueService.getPatientsByQueue(
      paramDto.type,
      queryDto.page,
      queryDto.limit,
      queryDto.classification,
      queryDto.nameFilter
    );

    return res
      .status(200)
      .json(
        SuccessResponse(
          enqueuedPatients,
          "Dados da fila capturado com sucesso!",
          200
        )
      );
  } catch (err: any) {
    next(err);
  }
}
}

```

ReportController.ts

```

import { NextFunction, Request, Response } from "express";
import { ReportService } from "../service/ReportService";
import { SuccessResponse } from "../utils/responses/SuccessResponse";
import { plainToInstance } from "class-transformer";
import { ReportParamsDto } from "../dtos/report/ReportParamsDto";

const reportService = new ReportService();

export class ReportController {
  static async getReport(req: Request, res: Response, next: NextFunction) {
    try {
      const params = plainToInstance(ReportParamsDto, req.query);

      const startDateObj = new Date(

```

```
    parseInt(params.startDate.split("-")[2]),
    parseInt(params.startDate.split("-")[1]) - 1,
    parseInt(params.startDate.split("-")[0]),
    0,
    0,
    0,
    0
);

const endDateObj = new Date(
    parseInt(params.endDate.split("-")[2]),
    parseInt(params.endDate.split("-")[1]) - 1,
    parseInt(params.endDate.split("-")[0]),
    23,
    59,
    59,
    999
);

const response = await reportService.getQueueReport(
    startDateObj,
    endDateObj
);

return res.json(
    SuccessResponse(response, "Dados coletados com sucesso", 200)
);
} catch (err) {
    next(err);
}
}
```

StatusController.ts

```
import { Request, Response } from "express";
import { ApiEnviroment } from "../enums/Api/ApiEnviroment";
import { config } from "dotenv";
import packageJson from "../../package.json";
import { ApiResponse } from "../utils/responses/ApiResponse";
import { SuccessResponse } from "../utils/responses/SuccessResponse";

config();
```

```

const API_NAME: string = packageJson.name;
const API_VERSION: string = "v" + (process.env.API_VERSION || "1");
const ENVIRONMENT: string = process.env.ENVIRONMENT || "dev";

const environment: ApiEnviroment =
  ENVIRONMENT === "prod" ? ApiEnviroment.PROD : ApiEnviroment.DEV;

class StatusController {
  static getStatus(req: Request, res: Response) {

    // Monta payload de status
    const response: ApiStatusResponse = {
      appName: API_NAME,
      version: API_VERSION,
      environment: environment,
    };

    // Retorna resposta de sucesso padronizada
    res.status(200).json(
      SuccessResponse(response, "API está online")
    );
  }
}

export default StatusController;

```

UserController.ts

```

import { plainToInstance } from "class-transformer";
import { UserDTO } from "../dtos/user/UserDto";
import { UserService } from "../service/UserService";
import { NextFunction, Request, Response } from "express";
import { SuccessResponse } from "../utils/responses/SuccessResponse";
import User from "../database/models/User";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { UserParamsDto } from "../dtos/user/UserParamsDto";

const userService = new UserService();

export class UserController {
  static async register(req: Request, res: Response, next: NextFunction) {
    try {
      const dto = plainToInstance(UserDTO, req.body);
      const { role, ...rest } = dto;

```

```

const user = await userService.createUser(rest);

return res
  .status(201)
  .json(SuccessResponse(user, "Usuário criado com sucesso!", 201));
} catch (err: any) {
  next(err);
}
}

static async updateName(req: Request, res: Response, next: NextFunction) {
try {
  const loggedInUser = req.user;
  if (!loggedInUser) throw new BadRequestError("Acesso negado");

  const dto = plainToInstance(UserDTO, req.body);

  await userService.changeName(loggedInUser.id, dto.name);
  res.status(200).json(SuccessResponse(null, "Nome atualizado", 200));
} catch (err) {
  next(err);
}
}

static async updateRole(req: Request, res: Response, next: NextFunction) {
try {
  const loggedInUser = req.user;
  if (!loggedInUser) throw new BadRequestError("Acesso negado");

  const paramDto = plainToInstance(UserParamsDto, req.params);
  const dto = plainToInstance(UserDTO, req.body);

  if (loggedInUser.id === paramDto.userId) {
    throw new BadRequestError(
      "Não é permitido alterar o próprio nível de acesso"
    );
  }

  await userService.updateRole(dto.role, paramDto.userId);
  res
    .status(200)
    .json(SuccessResponse(null, "Nível de acesso atualizado", 200));
} catch (err) {
  next(err);
}
}

```

```

    }
}

static async getUser(req: Request, res: Response, next: NextFunction) {
  try {
    const loggedInUser = req.user;
    if (!loggedInUser) throw new BadRequestError("Acesso negado");

    const user = await userService.getById(loggedInUser.id);
    res.status(200).json(SuccessResponse(user, "Usuário encontrado", 200));
  } catch (err) {
    next(err);
  }
}

static async deleteUser(req: Request, res: Response, next: NextFunction) {
  try {
    const loggedInUser = req.user;
    if (!loggedInUser) throw new BadRequestError("Acesso negado");

    await userService.deleteUser(loggedInUser as User);
    res.status(200).json(SuccessResponse(null, "Usuário deletado", 200));
  } catch (err) {
    next(err);
  }
}

```

Connection.ts

```

import { Options, Sequelize } from "sequelize";
import Logger from "../config/Logger";

let sequelize: Sequelize;

export const initDB = async (CONFIG: any): Promise<Sequelize> => {
  const { host, port, user, password, database } = CONFIG.mysql;

  const config: Options = {
    username: user,
    password,
    database,
    host,
    port,

```

```

    dialect: "mysql",
    logging: (msg) => Logger.debug(msg),
  };
  sequelize = new Sequelize(config);

  try {
    await sequelize.authenticate();
    Logger.info("Conexão com MySQL via Sequelize realizada com sucesso!");
  } catch (error) {
    Logger.error("Erro ao conectar ao MySQL:", error);
    throw error;
  }

  return sequelize;
};

export const getSequelize = (): Sequelize => {
  if (!sequelize) {
    throw new Error("Sequelize não foi inicializado. Chame initDB primeiro.");
  }
  return sequelize;
};

export const disconnectDB = async () => {
  if (!sequelize) return;
  try {
    await sequelize.close();
    Logger.warn("Conexão com MySQL encerrada!");
  } catch (error) {
    Logger.error("Erro ao fechar a conexão:", error);
  }
};

```

Attendance.ts

```

import { Model, DataTypes, Sequelize } from "sequelize";
import User from "./User";
import Patient from "./Patient";
import { ManchesterClassification } from "../../enums/Queue/ManchesterClassification";
import { Destination } from "../../enums/Attendance/Destination";
import { AttendanceStatus } from "../../enums/Attendance/AttendanceStatus";
import { IAttendance } from "../../interfaces/attendance/IAttendance";

export class Attendance extends Model {

```

```
declare id: number;
declare userId: number;
declare patientId: number;

declare entryDate: Date;
declare attendanceDate: Date;
declare dischargeDate: Date | null;

declare priority: ManchesterClassification;
declare mainComplaint: string;
declare currentIllnessHistory: string | null;
declare allergies: string | null;
declare currentMedications: string | null;

declare mainDiagnosis: string | null;
declare secondaryDiagnoses: string | null;
declare proceduresPerformed: string | null;

declare destination: Destination;
declare referralTo: string | null;
declare dischargeNotes: string | null;

declare status: AttendanceStatus;
declare createdAt: Date;
declare updatedAt: Date;

sanitize(): IAttendance {
  return {
    id: this.id,
    userId: this.userId,
    patientId: this.patientId,
    entryDate: this.entryDate,
    attendanceDate: this.attendanceDate,
    dischargeDate: this.dischargeDate,
    priority: this.priority,
    mainComplaint: this.mainComplaint,
    currentIllnessHistory: this.currentIllnessHistory,
    allergies: this.allergies,
    currentMedications: this.currentMedications,
    mainDiagnosis: this.mainDiagnosis,
    secondaryDiagnoses: this.secondaryDiagnoses,
```

```
proceduresPerformed: this.proceduresPerformed,  
  
destination: this.destination,  
referralTo: this.referralTo,  
dischargeNotes: this.dischargeNotes,  
  
status: this.status,  
};  
}  
  
static initialize(sequelize: Sequelize): void {  
    Attendance.init(  
    {  
        id: {  
            type: DataTypes.INTEGER.UNSIGNED,  
            autoIncrement: true,  
            primaryKey: true,  
        },  
        userId: {  
            type: DataTypes.INTEGER.UNSIGNED,  
            allowNull: false,  
            field: "usuario_id",  
        },  
        patientId: {  
            type: DataTypes.INTEGER.UNSIGNED,  
            allowNull: false,  
            field: "paciente_id",  
        },  
  
        entryDate: {  
            type: DataTypes.DATE,  
            allowNull: false,  
            field: "data_entrada",  
        },  
        attendanceDate: {  
            type: DataTypes.DATE,  
            allowNull: false,  
            field: "data_atendimento",  
        },  
        dischargeDate: {  
            type: DataTypes.DATE,  
            allowNull: true,  
            field: "data_alta",  
        },  
    },  
}
```

```
priority: {
    type: DataTypes.ENUM(...Object.values(ManchesterClassification)),
    allowNull: false,
    field: "prioridade",
},
mainComplaint: {
    type: DataTypes.STRING(500),
    allowNull: false,
    field: "queixa_principal",
},
currentIllnessHistory: {
    type: DataTypes.TEXT,
    allowNull: true,
    field: "historico_doenca_atual",
},
allergies: {
    type: DataTypes.TEXT,
    allowNull: true,
    field: "alergias",
},
currentMedications: {
    type: DataTypes.TEXT,
    allowNull: true,
    field: "medicamentos_uso",
},
mainDiagnosis: {
    type: DataTypes.STRING(500),
    allowNull: true,
    field: "diagnostico_principal",
},
secondaryDiagnoses: {
    type: DataTypes.TEXT,
    allowNull: true,
    field: "diagnosticos_secundarios",
},
proceduresPerformed: {
    type: DataTypes.TEXT,
    allowNull: true,
    field: "procedimentos_realizados",
},
destination: {
```

```

type: DataTypes.ENUM(...Object.values(Destination)),
allowNull: false,
defaultValue: Destination.DISCHARGE,
field: "destino",
},
referralTo: {
  type: DataTypes.STRING(300),
  allowNull: true,
  field: "encaminhamento_para",
},
dischargeNotes: {
  type: DataTypes.TEXT,
  allowNull: true,
  field: "observacoes_alta",
},
status: {
  type: DataTypes.ENUM(...Object.values(AttendanceStatus)),
  allowNull: false,
  defaultValue: AttendanceStatus.COMPLETED,
  field: "status",
},
},
{
  sequelize,
  tableName: "Atendimento",
  timestamps: true,
  createdAt: "data_criacao",
  updatedAt: "data_atualizacao",
}
);
}
}

static associate?(): void {
  Attendance.belongsTo(User, {
    foreignKey: "userId",
    as: "user",
  });
}

Attendance.belongsTo(Patient, {
  foreignKey: "patientId",
  as: "patient",
});

```

```

User.hasMany(Attendance, {
  foreignKey: "userId",
  as: "psAttendances",
});

Patient.hasMany(Attendance, {
  foreignKey: "patientId",
  as: "psAttendances",
});
}

export default Attendance;

```

index.ts

```

import { Sequelize } from "sequelize";
import fs from "fs";
import path from "path";
import { pathToFileURL } from "url";
import { IMModel } from "../../interfaces/IModel";

export const initializeModels = async (sequelize: Sequelize) => {
  const modelsPath = path.resolve(__dirname);
  const files = fs.readdirSync(modelsPath).filter((f) => f.endsWith(".ts"));

  const models: IMModel[] = [];

  for (const file of files) {
    if (file === "index.ts") continue;

    const modulePath = path.join(modelsPath, file);
    const moduleUrl = pathToFileURL(modulePath).href;

    const { [path.basename(file, path.extname(file))]: ModelClass } =
      await import(moduleUrl);

    if (ModelClass && typeof ModelClass.initialize === "function") {
      models.push(ModelClass);
    }
  }

  models.forEach((model) => model.initialize(sequelize));
  models.forEach((model) => model.associate?().);
}

```

```
};
```

Log.ts

```
import { DataTypes, Model, Sequelize } from "sequelize";
import { ILog } from "../../interfaces/log/ILog";

export class Log extends Model {
  declare id: number;
  declare userId: number;
  declare action: string;
  declare module: string;
  declare originIp: string | null;
  declare userAgent: string | null;

  declare createdAt: Date;

  sanitize(): ILog {
    return {
      id: this.id,
      userId: this.userId,
      action: this.action,
      module: this.module,
      originIp: this.originIp,
      userAgent: this.userAgent,
      createdAt: this.createdAt,
    };
  }
}

static initialize(sequelize: Sequelize): void {
  Log.init(
    {
      id: {
        type: DataTypes.INTEGER,
        autoIncrement: true,
        primaryKey: true,
      },
      userId: {
        type: DataTypes.INTEGER,
        allowNull: false,
        field: "usuario_id",
      },
      action: {
        type: DataTypes.STRING(100),
        allowNull: false,
        field: "acao",
      },
      module: {
        type: DataTypes.STRING(50),
      }
    }
  );
}
```

```

    allowNull: false,
    field: "modulo",
  },
  originIp: {
    type: DataTypes.STRING(45),
    allowNull: true,
    field: "ip_origem",
  },
  userAgent: {
    type: DataTypes.TEXT,
    allowNull: true,
    field: "user_agent",
  },
  createdAt: {
    type: DataTypes.DATE,
    defaultValue: DataTypes.NOW,
    field: "data_criacao",
  },
},
{
  sequelize,
  tableName: "registro",
  timestamps: false,
}
);
}
}

export default Log;

```

Manager.ts

```

import { DataTypes, Model, Sequelize } from "sequelize";
import User from "./User";
import { IManager } from "../../interfaces/manager/IManager";

export class Manager extends Model {
  declare id: number;
  declare userId: number;
  declare hospitalCode: string;

  declare data_criacao: Date;
  declare data_atualizacao: Date;

  sanitize(): IManager {
    return {
      id: this.id,
      userId: this.userId,
    }
  }
}

```

```

    hospitalCode: this.hospitalCode,
  };
}

static initialize(sequelize: Sequelize): void {
  Manager.init(
    {
      id: {
        type: DataTypes.INTEGER.UNSIGNED,
        autoIncrement: true,
        primaryKey: true,
      },
      userId: {
        type: DataTypes.INTEGER.UNSIGNED,
        allowNull: false,
        field: "usuario_id",
      },
      hospitalCode: {
        type: DataTypes.INTEGER.UNSIGNED,
        allowNull: false,
        field: "codigo_hospital",
      },
    },
    {
      sequelize,
      tableName: "gerente",
      timestamps: true,
      createdAt: "data_criacao",
      updatedAt: "data_atualizacao",
    }
  );
}

static associate(): void {
  Manager.belongsTo(User, { foreignKey: "usuario_id", as: "user" });
  User.hasOne(Manager, { foreignKey: "usuario_id", as: "manager" });
}

export default Manager;

```

Patient.ts

```

import { DataTypes, Model, Sequelize } from "sequelize";
import { Gender } from "../../enums/User/Gender";

```

```
import User from "./User";
import { IPatient } from "../../interfaces/patient/IPatient";
import { calculateAge } from "../../utils/CalculateUtils";

export class Patient extends Model {
    declare id: number;
    declare userId: number;
    declare cpf: string;
    declare gender: Gender;
    declare phone: string;
    declare birthDate: Date;

    declare cnsNumber: string;
    declare icePhone: string;
    declare motherName: string;
    declare user?: User;

    declare data_criacao: Date;
    declare data_atualizacao: Date;

    sanitize(): IPatient {
        return {
            id: this.id,
            userId: this.userId,
            cpf: this.cpf,
            gender: this.gender,
            phone: this.phone,
            birthDate: this.birthDate,
            cnsNumber: this.cnsNumber,
            icePhone: this.icePhone,
            motherName: this.motherName,
            age: calculateAge(this.birthDate),
        };
    }
}

static initialize(sequelize: Sequelize): void {
    Patient.init(
        {
            id: {
                type: DataTypes.INTEGER.UNSIGNED,
                autoIncrement: true,
                primaryKey: true,
            },
            userId: {

```

```
type: DataTypes.INTEGER,
allowNull: false,
unique: true,
field: "usuario_id",
},
cpf: {
  type: DataTypes.STRING(14), // formato: 000.000.000-00
  allowNull: true,
  unique: true,
},
phone: {
  type: DataTypes.STRING(20), // formato: (00) 00000-0000
  allowNull: true,
  field: "telefone",
},
gender: {
  type: DataTypes.ENUM(...Object.values(Gender)),
  allowNull: true,
  field: "sexo",
},
cnsNumber: {
  type: DataTypes.STRING(20),
  allowNull: true,
  field: "numero_cns",
},
icePhone: {
  type: DataTypes.STRING(20),
  allowNull: true,
  field: "telefone_emergencia",
},
motherName: {
  type: DataTypes.STRING(100),
  allowNull: true,
  field: "nome_mae",
},
birthDate: {
  type: DataTypes.DATEONLY,
  allowNull: true,
  field: "data_nascimento",
},
},
{
  sequelize,
  tableName: "paciente",
```

```

    timestamps: true,
    createdAt: "data_criacao",
    updatedAt: "data_atualizacao",
  }
);
}
}

static associate(): void {
  Patient.belongsTo(User, { foreignKey: "usuario_id", as: "user" });
  UserhasOne(Patient, { foreignKey: "usuario_id", as: "patient" });
}
}

export default Patient;

```

QueueEvent.ts

```

import { DataTypes, Model, Sequelize } from "sequelize";
import { IQueueEvent } from "../../interfaces/queue/IQueueEvent";

export class QueueEvent extends Model {
  declare id: string;
  declare patientId: string;
  declare queue: "triagem" | "atendimento";
  declare enteredAt: Date;
  declare startedAt: Date | null;
  declare finishedAt: Date | null;
  declare queueId: string;
  declare createdAt: Date;
  declare updatedAt: Date;

  sanitize(): IQueueEvent {
    return {
      id: this.id,
      patientId: this.patientId,
      queue: this.queue,
      enteredAt: this.enteredAt,
      startedAt: this.startedAt,
      finishedAt: this.finishedAt,
      queueId: this.queueId,
    };
  }
}

static initialize(sequelize: Sequelize): void {
  QueueEvent.init(

```

```
{  
  id: {  
    type: DataTypes.INTEGER,  
    autoIncrement: true,  
    primaryKey: true,  
  },  
  patientId: {  
    type: DataTypes.INTEGER,  
    allowNull: false,  
    field: "paciente_id",  
  },  
  queueId: {  
    type: DataTypes.STRING,  
    allowNull: false,  
    field: "fila_id",  
  },  
  queue: {  
    type: DataTypes.ENUM("triagem", "atendimento"),  
    allowNull: false,  
    field: "fila",  
  },  
  enteredAt: {  
    type: DataTypes.DATE,  
    allowNull: false,  
    field: "entrou_em",  
  },  
  startedAt: {  
    type: DataTypes.DATE,  
    allowNull: true,  
    field: "iniciou_em",  
  },  
  {  
    sequelize,  
    tableName: "fila_eventos",  
    timestamps: false,  
  }  
};  
}  
  
export default QueueEvent;
```

RefreshToken.ts

```

import { Model, DataTypes, Sequelize } from "sequelize";
import User from "./User";

export class RefreshToken extends Model {
  declare id: number;
  declare userId: number;
  declare token: string;

  declare data_criacao: Date;
  declare data_atualizacao: Date;

  static initialize(sequelize: Sequelize): void {
    RefreshToken.init(
      {
        id: {
          type: DataTypes.INTEGER.UNSIGNED,
          autoIncrement: true,
          primaryKey: true,
        },
        userId: {
          type: DataTypes.INTEGER.UNSIGNED,
          allowNull: false,
          field: "usuario_id",
        },
        token: {
          type: DataTypes.STRING,
          allowNull: false,
        },
      },
      {
        sequelize,
        tableName: "refresh_tokens",
        timestamps: true,
        createdAt: "data_criacao",
        updatedAt: "data_atualizacao",
      }
    );
  }

  static associate(): void {
    User.hasMany(RefreshToken, { foreignKey: "userId", as: "refreshTokens" });
    RefreshToken.belongsTo(User, { foreignKey: "userId", as: "user" });
  }
}

```

```
}
```

User.ts

```
import { DataTypes, Model, Sequelize } from "sequelize";
import { IUser } from "../../interfaces/user/IUser";

export class User extends Model {
  declare id: number;
  declare name: string;
  declare email: string;
  declare password: string;
  declare role: number;
  declare deleted: boolean;

  declare data_criacao: Date;
  declare data_atualizacao: Date;

  sanitize(): IUser {
    return {
      id: this.id,
      name: this.name,
      email: this.email,
      role: this.role,
    };
  }
}

static initialize(sequelize: Sequelize): void {
  User.init(
    {
      id: {
        type: DataTypes.INTEGER.UNSIGNED,
        autoIncrement: true,
        primaryKey: true,
      },
      name: {
        type: DataTypes.STRING(100),
        allowNull: false,
        field: "nome",
      },
      email: {
        type: DataTypes.STRING(150),
        allowNull: false,
        unique: true,
      },
    }
  );
}
```

```

validate: {
  isEmail: true,
},
},
deleted: {
  type: DataTypes.BOOLEAN,
  defaultValue: false,
  field: "deletado",
},
password: {
  type: DataTypes.STRING(255),
  allowNull: false,
  field: "senha",
},
role: {
  type: DataTypes.INTEGER,
  allowNull: false,
  defaultValue: 1,
  field: "nivel_acesso",
},
},
{
  sequelize,
  tableName: "usuario",
  timestamps: true,
  createdAt: "data_criacao",
  updatedAt: "data_atualizacao",
}
);
}
}

export default User;

```

DateRange.ts

```

import {
  ValidationArguments,
  ValidatorConstraintInterface,
} from "class-validator";

export class IsStartDateBeforeEndDateConstraint
  implements ValidatorConstraintInterface
{

```

```

validate(endDate: string, args: ValidationArguments) {
  const startDate = (args.object as any).startDate;

  const [startDay, startMonth, startYear] = startDate.split("-").map(Number);
  const [endDay, endMonth, endYear] = endDate.split("-").map(Number);

  const start = new Date(startYear, startMonth - 1, startDay);
  const end = new Date(endYear, endMonth - 1, endDay);

  return start <= end;
}

defaultMessage(args: ValidationArguments) {
  return "A data de início não pode ser maior que a data de fim";
}
}

```

Match.ts

```

import {
  registerDecorator,
  ValidationOptions,
  ValidationArguments,
} from "class-validator";

export function Match(property: string, validationOptions?: ValidationOptions) {
  return function (object: Object, propertyName: string) {
    registerDecorator({
      name: "match",
      target: object.constructor,
      propertyName: propertyName,
      options: validationOptions,
      constraints: [property],
      validator: {
        validate(value: any, args: ValidationArguments) {
          const [relatedPropertyName] = args.constraints;
          const relatedValue = (args.object as any)[relatedPropertyName];
          return value === relatedValue;
        },
        defaultMessage(args: ValidationArguments) {
          const [relatedPropertyName] = args.constraints;
          return `${propertyName} deve ser igual a ${relatedPropertyName}`;
        },
      },
    });
  };
}

```

```

    });
};

}

```

PasswordRequirements.ts

```

import {
  registerDecorator,
  ValidationArguments,
  ValidationOptions,
} from "class-validator";

export function PasswordRequirements(validationOptions?: ValidationOptions) {
  return function (object: Object, propertyName: string) {
    registerDecorator({
      name: "PasswordRequirements",
      target: object.constructor,
      propertyName,
      options: validationOptions,
      validator: {
        validate(value: string) {
          if (typeof value !== "string") return false;
          return (
            value.length >= 8 &&
            /[a-z]/.test(value) &&
            /[A-Z]/.test(value) &&
            /\d/.test(value) &&
            /[^A-Za-z0-9]/.test(value)
          );
        },
        defaultMessage(args: ValidationArguments) {
          const value = args.value as string;
          const missing: string[] = [];

          if (!value || value.length < 8) missing.push("8 caracteres");
          if (!/[a-z]/.test(value)) missing.push("letra minúscula");
          if (!/[A-Z]/.test(value)) missing.push("letra maiúscula");
          if (!/\d/.test(value)) missing.push("número");
          if (!/[^A-Za-z0-9]/.test(value)) missing.push("símbolo");

          return `A senha deve conter: ${missing.join(", ")}.`;
        },
      });
    });
  };
}

```

```
    };
}
```

AttendanceDto.ts

```
import { Type } from "class-transformer";
import {
  IsDate,
  IsEnum,
  IsOptional,
  IsString,
} from "class-validator";
import { Destination } from "../../enums/Attendance/Destination";
import { AttendanceStatus } from "../../enums/Attendance/AttendanceStatus";

export class AttendanceDto {
  /**
   * Data de alta
   */
  @IsOptional({ groups: ["update"] })
  @Type(() => Date)
  @IsDate({
    message: "A Data de alta deve ser uma data válida",
    groups: ["update"],
  })
  dischargeDate?: Date | null;

  /**
   * Queixa principal
   */
  @IsOptional({ groups: ["update"] })
  @IsString({
    message: "A Queixa principal deve ser um texto",
    groups: ["update"],
  })
  mainComplaint: string;

  /**
   * Histórico da doença atual
   */
  @IsOptional({ groups: ["update"] })
  @IsString({
    message: "O Histórico da doença atual deve ser um texto",
    groups: ["update"],
  })
}
```

```

        })
    currentIllnessHistory?: string | null;

    /**
     * Alergias
     */
    @IsOptional({ groups: ["update"] })
    @IsString({
        message: "As Alergias devem ser um texto",
        groups: ["update"],
    })
    allergies?: string | null;

    /**
     * Medicamentos atuais
     */
    @IsOptional({ groups: ["update"] })
    @IsString({
        message: "Os Medicamentos atuais devem ser um texto",
        groups: ["update"],
    })
    currentMedications?: string | null;

    /**
     * Diagnóstico principal
     */
    @IsOptional({ groups: ["update"] })
    @IsString({
        message: "O Diagnóstico principal deve ser um texto",
        groups: ["update"],
    })
    mainDiagnosis?: string | null;

    /**
     * Diagnósticos secundários
     */
    @IsOptional({ groups: ["update"] })
    @IsString({
        message: "Os Diagnósticos secundários devem ser um texto",
        groups: ["update"],
    })
    secondaryDiagnoses?: string | null;
}

```

```
* Procedimentos realizados
*/
@IsOptional({ groups: ["update"] })
@IsString({
  message: "Os Procedimentos realizados devem ser um texto",
  groups: ["update"],
})
proceduresPerformed?: string | null;

/***
 * Destino
 */
@IsOptional({ groups: ["update"] })
@IsEnum(Destination, {
  message: "O Destino deve ser um destino válido",
  groups: ["update"],
})
destination: Destination;

/***
 * Encaminhamento para
 */
@IsOptional({ groups: ["update"] })
@IsString({
  message: "O Encaminhamento para deve ser um texto",
  groups: ["update"],
})
referralTo?: string | null;

/***
 * Notas de alta
 */
@IsOptional({ groups: ["update"] })
@IsString({
  message: "As Notas de alta devem ser um texto",
  groups: ["update"],
})
dischargeNotes?: string | null;

/***
 * Status do atendimento
 */
@IsOptional({ groups: ["update"] })
@IsEnum(AttendanceStatus, {
```

```

    message: "O status deve ser um status de atendimento válido",
    groups: ["update"],
  })
  status: AttendanceStatus;
}

```

AttendanceParamDto.ts

```

import { Type } from "class-transformer";
import { IsNumber, Min } from "class-validator";

export class AttendanceParamDto {
  /**
   * ID do paciente
   */
  @Type(() => Number)
  @IsNumber(
    {},
    {
      message: "O id do paciente deve ser um número",
      groups: ["search"],
    }
  )
  @Min(1, {
    message: "O id do paciente deve ser pelo menos 1",
    groups: ["search"],
  })
  patientId: number;

  /**
   * ID do atendimento
   */
  @Type(() => Number)
  @IsNumber(
    {},
    {
      message: "O id do atendimento deve ser um número",
      groups: ["delete", "update", "get"],
    }
  )
  @Min(1, {
    message: "O id do atendimento deve ser pelo menos 1",
    groups: ["delete", "update", "get"],
  })
}

```

```
attendanceId: number;
}
```

AttendanceQueryDto.ts

```
import { Type } from "class-transformer";
import { IsNumber, Min } from "class-validator";

export class AttendanceQueryDto {
    /**
     * Página inicial da busca (0-based)
     */
    @Type(() => Number)
    @IsNumber(
        {},
        { message: "O parâmetro 'page' deve ser um número", groups: ["search"] }
    )
    @Min(0, {
        message: "O parâmetro 'page' não pode ser negativo",
        groups: ["search"],
    })
    page: number = 0;

    /**
     * Quantidade máxima de itens por página
     */
    @Type(() => Number)
    @IsNumber(
        {},
        { message: "O parâmetro 'limit' deve ser um número", groups: ["search"] }
    )
    @Min(1, {
        message: "O parâmetro 'limit' deve ser pelo menos 1",
        groups: ["search"],
    })
    limit: number = 10;
}
```

EmployeeDto.ts

```
import { Type } from "class-transformer";
import { IsNumber, IsOptional, IsString, Min } from "class-validator";
```

```

export class EmployeeDto {
    /**
     * Página inicial da busca (0-based)
     */
    @Type(() => Number)
    @IsNumber(
        {},
        { message: "O parâmetro 'page' deve ser um número", groups: ["search"] }
    )
    @Min(0, {
        message: "O parâmetro 'page' não pode ser negativo",
        groups: ["search"],
    })
    page: number = 0;

    /**
     * Quantidade máxima de itens por página
     */
    @Type(() => Number)
    @IsNumber(
        {},
        { message: "O parâmetro 'limit' deve ser um número", groups: ["search"] }
    )
    @Min(1, {
        message: "O parâmetro 'limit' deve ser pelo menos 1",
        groups: ["search"],
    })
    limit: number = 10;

    /**
     * Nome do paciente para ser buscado
     */
    @IsOptional({ groups: ["search"] })
    @IsString({
        message: "O filtro do nome deve ser um texto",
        groups: ["search"],
    })
    nameFilter: string;
}

```

LogFilterDto.ts

```

import {
    IsNumber,

```

```

IsString,
IsOptional,
Min,
IsPositive,
} from "class-validator";
import { Type } from "class-transformer";

export class LogFilterDto {
  @IsOptional()
  @Type(() => Number)
  @IsNumber()
  @Min(0)
  page?: number = 0;

  @IsOptional()
  @Type(() => Number)
  @IsNumber()
  @IsPositive()
  @Min(1)
  limit?: number = 10;

  @IsOptional()
  @Type(() => Number)
  @IsNumber()
  @IsPositive()
  userId?: number;

  @IsOptional()
  @IsString()
  action?: string;

  @IsOptional()
  @IsString()
  module?: string;
}

```

ManagerDto.ts

```

import { Type } from "class-transformer";
import { IsNumber, IsOptional, IsString, Min } from "class-validator";

export class ManagerDto {
  /**
   * Página inicial da busca (0-based)

```

```

*/
@Type(() => Number)
@IsNumber(
  {},
  { message: "O parâmetro 'page' deve ser um número", groups: ["search"] }
)
@Min(0, {
  message: "O parâmetro 'page' não pode ser negativo",
  groups: ["search"],
})
page: number = 0;

/**
 * Quantidade máxima de itens por página
 */
@Type(() => Number)
@IsNumber(
  {},
  { message: "O parâmetro 'limit' deve ser um número", groups: ["search"] }
)
@Min(1, {
  message: "O parâmetro 'limit' deve ser pelo menos 1",
  groups: ["search"],
})
limit: number = 10;

/**
 * Nome do paciente para ser buscado
 */
@IsOptional({ groups: ["search"] })
@IsString({
  message: "O filtro do nome deve ser um texto",
  groups: ["search"],
})
nameFilter: string;
}

```

PatientDto.ts

```

import {
  IsString,
  IsOptional,
  IsEnum,
  IsDateString,

```

```

IsPhoneNumber,
Matches,
Length,
MaxLength,
MinLength,
} from "class-validator";
import { Gender } from "../../enums/User/Gender";

export class PatientDto {
  @IsString({
    message: "O nome deve ser um texto",
    groups: ["update"],
  })
  @Length(2, 100, {
    message: "O nome deve ter entre 2 e 100 caracteres",
    groups: ["update"],
  })
  @IsOptional({ groups: ["update"] })
  name?: string;

  @IsString({
    message: "O CPF deve ser uma string",
    groups: ["create", "update"],
  })
  @Matches(/^\d{11}$/, {
    message: "CPF deve conter exatamente 11 dígitos",
    groups: ["create", "update"],
  })
  @IsOptional({ groups: ["update"] })
  cpf?: string;

  @IsPhoneNumber("BR", {
    message: "O telefone deve ser um número brasileiro válido",
    groups: ["create", "update"],
  })
  @IsOptional({ groups: ["create", "update"] })
  phone?: string;

  @IsEnum(Gender, { message: "Sexo inválido" })
  @IsOptional({ groups: ["update"] })
  gender?: Gender;

  @IsDateString(
    {},
  )
}

```

```

{
  message: "A data de nascimento deve ser uma data válida",
  groups: ["create", "update"],
}
)
@IsOptional({ groups: ["update"] })
birthDate?: string;

@IsString({
  message: "CNS deve ser uma string",
  groups: ["create", "update"],
})
@Matches(/^\d{15}$/, {
  message: "CNS deve conter exatamente 15 dígitos numéricos",
  groups: ["create", "update"],
})
@IsOptional({ groups: ["update"] })
cnsNumber?: string;

@IsPhoneNumber("BR", {
  message: "O telefone de emergência deve ser um número de telefone válido",
  groups: ["create", "update"],
})
@IsOptional({ groups: ["update"] })
icePhone?: string;

@MaxLength(100, {
  message: "O nome da mãe deve ter no máximo 100 caracteres",
  groups: ["create", "update"],
})
@MinLength(3, {
  message: "O nome da mãe deve ter no mínimo 3 caracteres",
  groups: ["create", "update"],
})
@IsOptional({ groups: ["update"] })
motherName?: string;
}

```

EnterQueueDto.ts

```

import { IsString } from "class-validator";

export class EnterQueueDto {
  @IsString({

```

```

    message: "O código do hospital deve ser um texto",
    groups: ["search"],
  })
  hospitalCode: string;
}

```

QueueDto.ts

```

import { Type } from "class-transformer";
import { IsEnum, IsNotEmpty, IsNumber, IsOptional, IsString, Min } from "class-validator";
import { ManchesterClassification } from "../../enums/Queue/ManchesterClassification";

export class QueueDto {
  /**
   * Página inicial da busca (0-based)
   */
  @Type(() => Number)
  @IsNumber(
    {},
    { message: "O parâmetro 'page' deve ser um número", groups: ["search"] }
  )
  @Min(0, {
    message: "O parâmetro 'page' não pode ser negativo",
    groups: ["search"],
  })
  page: number = 0;

  /**
   * Quantidade máxima de itens por página
   */
  @Type(() => Number)
  @IsNumber(
    {},
    { message: "O parâmetro 'limit' deve ser um número", groups: ["search"] }
  )
  @Min(1, {
    message: "O parâmetro 'limit' deve ser pelo menos 1",
    groups: ["search"],
  })
  limit: number = 10;

  /**
   * Classificação de risco do paciente (triagem Manchester)
   * Valores válidos: 'imediato', 'muito urgente', 'urgente', 'pouco urgente', 'não urgente'
  */
}

```

```

*/
@IsOptional({ groups: ["search"] })
@IsEnum(ManchesterClassification, {
  message:
    "A classificação informada não é válida. Valores válidos: imediato, muito urgente,
urgente, pouco urgente, não urgente",
  groups: ["next", "search"],
})
classification: ManchesterClassification;

/***
 * Nome do paciente para ser buscado
 */
@IsOptional({ groups: ["search"] })
@IsString({
  message: "O filtro do nome deve ser um texto",
  groups: ["search"],
})
nameFilter: string;

/***
 * Nome da sala para o paciente ir
 */
@IsString({
  message: "A sala deve ser um texto",
  groups: ["next-patient"],
})
@IsNotEmpty({
  message: "A sala não pode ser vazia",
  groups: ["next-patient"],
})
room: string;
}

```

QueueParamsDto.ts

```

import { IsEnum, IsNumber, Min } from "class-validator";
import { QueueType } from "../../enums/Queue/QueueType";
import { Type } from "class-transformer";

export class QueueParamsDto {
  /**
   * Tipo da fila de atendimento
   * Valores válidos: 'triage' ou 'treatment'

```

```

*/
@IsEnum(QueueType, {
  message:
    "O tipo da fila informado não é válido. Deve ser 'triage' ou 'treatment'.",
  groups: ["search"],
})
type: QueueType;

/**
 * ID do paciente
 */
@Type(() => Number)
@IsNumber(
  {},
  { message: "O parâmetro 'patientId' deve ser um número", groups: ["next"] }
)
@Min(1, {
  message: "O 'patientId' deve ser maior que zero",
  groups: ["next"],
})
patientId: number;
}

```

ReportParamsDto.ts

```

import { Matches, IsString, Validate } from "class-validator";
import { IsStartDateBeforeEndDateConstraint } from "../../decorators/DateRange";

export class ReportParamsDto {
  /**
   * Data de início para o relatório (formato DD-MM-YYYY)
   * Exemplo: 25-10-2005
   */
  @IsString({ message: "A data de início deve ser uma string" })
  @Matches(/^(0[1-9]|1[0-2])-(0[1-9]|1[0-2])-([0-9]{4})$/, {
    message:
      "A data de início deve estar no formato DD-MM-YYYY (ex: 25-10-2005)",
  })
  startDate: string;

  /**
   * Data de fim para o relatório (formato DD-MM-YYYY)
   * Exemplo: 25-10-2005
   */

```

```

@IsString({ message: "A data de fim deve ser uma string" })
@Matches(/^(0[1-9][12][0-9]3[01])-(0[1-9]1[0-2])-\d{4}$/, {
  message: "A data de fim deve estar no formato DD-MM-YYYY (ex: 25-10-2005)",
})
@Validate(IsStartDateBeforeEndDateConstraint)
endDate: string;
}

```

UserDto.ts

```

import {
  IsString,
  IsEmail,
  Length,
  IsIn,
  IsStrongPassword,
} from "class-validator";
import { Match } from "../../decorators/Match";
import { UserRole } from "../../enums/User/UserRole";
import { Type } from "class-transformer";
import { PasswordRequirements } from "../../decorators>PasswordRequirements";

export class UserDTO {
  /**
   * Nome do usuário
   * Deve ter entre 2 e 100 caracteres
   */
  @IsString({
    message: "O nome deve ser um texto",
    groups: ["create", "update", "updateName"],
  })
  @Length(2, 100, {
    message: "O nome deve ter entre 2 e 100 caracteres",
    groups: ["create", "update", "updateName"],
  })
  name!: string;

  /**
   * Email do usuário
   */
  @IsEmail(
    {},
    {
      message: "O email informado não é válido",
    }
  )
}

```

```

        groups: ["create", "login", "forgot", "changeEmail"],
    }
)
email!: string;

/***
 * Senha do usuário
 * Deve ter no mínimo 8 caracteres
 */
@IsString({
    message: "A senha deve ser um texto",
    groups: ["create", "login", "reset"],
})
@PasswordRequirements({
    groups: ["create", "reset"],
})
password!: string;

/***
 * Confirmação da senha
 * Deve ser igual à senha
 */
@IsString({
    message: "A confirmação da senha deve ser um texto",
    groups: ["create", "reset"],
})
@Match("password", {
    message: "As senhas não conferem",
    groups: ["create", "reset"],
})
confirmPassword!: string;

@Type(() => Number)
@IsIn([UserRole.ADMIN, UserRole.EMPLOYEE], {
    message: `Role inválida. Use ${UserRole.ADMIN} (ADMIN) ou
${UserRole.EMPLOYEE} (EMPLOYEE).`,
    groups: ["role"],
})
role!: UserRole;
}

```

UserParamsDto.ts

```
import { IsNumber, Min } from "class-validator";
```

```

import { Type } from "class-transformer";

export class UserParamsDto {
    /**
     * Id do usuário
     */
    @Type(() => Number)
    @IsNumber(
        {},
        { message: "O parâmetro 'userId' deve ser um número", groups: ["role"] }
    )
    @Min(0, {
        message: "O 'userId' deve ser maior que zero",
        groups: ["role"],
    })
    userId: number;
}

```

ApiEnviroment.ts

```

export enum ApiEnviroment {
    DEV = "development",
    PROD = "production",
}

```

ApiStatus.ts

```

export enum ApiStatus {
    ONLINE = "online",
    MAINTENANCE = "maintenance",
    OFFLINE = "offline",
}

```

AttendanceStatus.ts

```

export enum AttendanceStatus {
    DRAFT = "rascunho",
    COMPLETED = "finalizado",
    CANCELLED = "cancelado",
}

```

Destination.ts

```
export enum Destination {
  DISCHARGE = "alta",
  HOSPITALIZATION = "internacao",
  OBSERVATION = "observacao",
  TRANSFER = "transferencia",
  DEATH = "obito",
  ABSCONDED = "fugiu"
}
```

ManchesterClassification.ts

```
export enum ManchesterClassification {
  RED = "imediato", // Emergência – atendimento imediato
  ORANGE = "muito urgente", // Muito urgente – atendimento em até 10 min
  YELLOW = "urgente", // Urgente – atendimento em até 60 min
  GREEN = "pouco urgente", // Pouco urgente – atendimento em até 120 min
  BLUE = "não urgente", // Não urgente – atendimento em até 240 min
}
```

QueueStatus.ts

```
export enum QueueStatus {
  WAITING = "WAITING", // aguardando na fila
  IN_PROGRESS = "IN_PROGRESS", // chamado, mas ainda em atendimento
}
```

QueueType.ts

```
export enum QueueType {
  TRIAGE = "triage",
  TREATMENT = "treatment",
}
```

QueueTypePT.ts

```
import { QueueType } from "./QueueType"

export const QueueTypePT: Record<QueueType, string> = {
  triage: "Triagem",
  treatment: "Atendimento",
```

```
};
```

TokenTypes.ts

```
export enum TokenType {
  AUTH = "AUTH",
  GENERATE_ACCOUNT = "GEN_ACCOUNT",
  RESET_PASS = "RESET_PASS",
  RESET_EMAIL = "RESET_EMAIL"
}
```

Gender.ts

```
export enum Gender {
  MALE = 'Masculino',
  FEMALE = 'Feminino',
}
```

UserRole.ts

```
export enum UserRole {
  PATIENT = 1,
  ADMIN = 0,
  EMPLOYEE = 2,
}
```

NotificationGateway.ts

```
import { SocketServer } from "../config/SocketServer";
import Logger from "../config/Logger";

export class NotificationGateway {
  static emitToUser(userId: number, event: string, data: any) {
    try {
      SocketServer.getIo().to(`user:${userId}`).emit(event, data);
    } catch (err: any) {
      Logger.error("Falha ao emitir notificação:", err);
    }
  }

  static emitToAdmins(event: string, data: any) {
    try {
      SocketServer.getIo().to("admin:panel").emit(event, data);
    } catch (err: any) {
      Logger.error("Falha ao emitir para admins:", err);
    }
  }
}
```

```

        }
    }

    static isUserOnline(userId: number): boolean {
        try {
            const room = SocketServer.getIo().sockets.adapter.rooms.get(
                `user:${userId}`
            );
            return !!room && room.size > 0;
        } catch (err: any) {
            return false;
        }
    }
}

```

IErrorDetail.ts

```
export interface IErrorDetail {
    field: string;
    message: string;
}
```

IGenericError.ts

```
export interface IGenericError {
    status?: number;
    message: string;
}
```

IModel.ts

```
import { Sequelize } from "sequelize";

export interface IMModel {
    initialize(sequelize: Sequelize): void;
    associate?(): void;
}
```

ITokenPayload.ts

```
import { JwtPayload } from "jsonwebtoken";

export interface ITokenPayload extends JwtPayload {
    id: number;
}
```

```
role: number;
email?: string;
hospitalCode?: string;
}
```

IAttendance.ts

```
import { AttendanceStatus } from "../../enums/Attendance/AttendanceStatus";
import { Destination } from "../../enums/Attendance/Destination";
import { ManchesterClassification } from "../../enums/Queue/ManchesterClassification";

export interface IAttendance {
  id: number;
  userId: number;
  patientId: number;

  entryDate: Date;
  attendanceDate: Date;
  dischargeDate: Date | null;

  priority: ManchesterClassification;
  mainComplaint: string;
  currentIllnessHistory: string | null;
  allergies: string | null;
  currentMedications: string | null;

  mainDiagnosis: string | null;
  secondaryDiagnoses: string | null;
  proceduresPerformed: string | null;

  destination: Destination;
  referralTo: string | null;
  dischargeNotes: string | null;

  status: AttendanceStatus;
}
```

IAttendanceQueryParams.ts

```
export interface IAttendanceQueryParams {
  page?: number;
  limit?: number;
  patientId?: number;
```

```

    startDate?: Date;
    endDate?: Date;
}

```

ICreateAttendanceInput.ts

```

import { AttendanceStatus } from "../../enums/Attendance/AttendanceStatus";
import { Destination } from "../../enums/Attendance/Destination";
import { ManchesterClassification } from "../../enums/Queue/ManchesterClassification";

export interface ICreateAttendanceInput {
    userId: number;
    patientId: number;
    entryDate: Date;
    attendanceDate: Date;
    dischargeDate?: Date | null;
    priority: ManchesterClassification;
    mainComplaint: string;
    currentIllnessHistory?: string | null;
    allergies?: string | null;
    currentMedications?: string | null;
    mainDiagnosis?: string | null;
    secondaryDiagnoses?: string | null;
    proceduresPerformed?: string | null;
    destination: Destination;
    referralTo?: string | null;
    dischargeNotes?: string | null;
    status: AttendanceStatus;
}

```

IUpdateAttendanceInput.ts

```

import { AttendanceStatus } from "../../enums/Attendance/AttendanceStatus";
import { Destination } from "../../enums/Attendance/Destination";

export interface IUpdateAttendanceInput {
    dischargeDate?: Date | null;
    mainComplaint: string;
    currentIllnessHistory?: string | null;
    allergies?: string | null;
    currentMedications?: string | null;
    mainDiagnosis?: string | null;
    secondaryDiagnoses?: string | null;

```

```

proceduresPerformed?: string | null;
destination: Destination;
referralTo?: string | null;
dischargeNotes?: string | null;
status: AttendanceStatus;
}

```

IFieldError.ts

```

export interface IFieldError {
  field: string;
  message: string;
}

```

FieldError.ts

```

export interface FieldError {
  field: string;
  message: string;
}

```

ILog.ts

```

export interface ILog {
  id: number;
  userId: number;
  action: string;
  module: string;
  originIp: string | null;
  userAgent: string | null;
  createdAt: Date;
}

```

ILogCreate.ts

```

export interface ILogCreate {
  userId: number;
  action: string;
  module: string;
  originIp?: string | null;
  userAgent?: string | null;
}

```

ILogFilter.ts

```
export interface ILogFilter {
  page?: number;
  limit: number;
  offset?: number;
  userId?: number;
  action?: string;
  module?: string;
}
```

ILoggerOptions.ts

```
export interface ILoggerOptions {
  action: string;
  resource: string;
}
```

ICreateManagerInput.ts

```
export interface ICreateManagerInput {
  userId: number;
  hospitalCode: string;
}
```

IManager.ts

```
export interface IManger {
  id: number;
  userId: number;
  hospitalCode: string;
}
```

IPatient.ts

```
import { Gender } from "../../enums/User/Gender";

export interface IPatient {
  id: number;
  userId: number;
  cpf: string;
  gender: Gender;
  phone: string;
  birthDate: Date;
  age: number;
```

```
cnsNumber: string;
icePhone: string;
motherName: string;
}
```

ICreateQueueEventInput.ts

```
export interface IAvgWait {
  avgWaitSeconds: number;
}

export interface ICreateQueueEventInput {
  patientId: number;
  queueId: string;
  enteredAt: Date;
  startedAt?: Date;
}
```

IPagination.ts

```
export interface IPagination {
  currentPage: number;
  totalPages: number;
  totalItems: number;
  itemsPerPage: number;
  hasNext: boolean;
  hasPrev: boolean;
  nextPage: number | null;
  prevPage: number | null;
}
```

IQueuedPatient.ts

```
import { ManchesterClassification } from "../../enums/Queue/ManchesterClassification";
import { Gender } from "../../enums/User/Gender";

export interface IQueuedPatient {
  id: number;
  name: string;
  age: number;
  birthDate: Date;
  cnsNumber: string;
```

```

motherName: string;
dateHourAttendance: string;
gender: Gender;
position: number;
attendanceId?: number;
classification: ManchesterClassification | null;
}

```

IQueueEvent.ts

```

export interface IQueueEvent {
  id: string;
  patientId: string;
  queue: "triagem" | "atendimento";
  enteredAt: Date;
  startedAt: Date | null;
  finishedAt: Date | null;
  queueId: string;
}

```

IQueueHistoryResponse.ts

```

export interface IQueueHistoryResponse {
  name: string;
  room: string;
  timestamp: number;
}

```

IReportResponse.ts

```

export interface IReportResponse {
  avgTimeTreatmentInSec: number[];
  avgTimeTriageInSec: { averageWaitTime: number; count: number }[];
  peakDemand: { [key: string]: number };
}

```

ICreateUser.ts

```

export interface ICreateUserInput {
  name: string;
  email: string;
  password: string;
  role?: number;
}

```

```
}
```

IUser.ts

```
export interface IUser {
  id: number;
  name: string;
  email: string;
  role: number;
}
```

IUserQueryParams.ts

```
import { UserRole } from "../../enums/User/UserRole";

export interface IUserQueryParams {
  page?: number;
  limit?: number;
  name?: string;
  role: UserRole;
}
```

AuthMiddleware.ts

```
import { Request, Response, NextFunction } from "express";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { TokenUtils } from "../utils/TokenUtils";
import { TokenType } from "../enums/Token/TokenType";
import { ForbiddenError } from "../utils/errors/ForbiddenError";

var tokenUtils = new TokenUtils();

export async function AuthMiddleware(
  req: Request,
  res: Response,
  next: NextFunction
) {
  const token = getBearerToken(req);
  if (!token) return next(new BadRequestError("Token ausente"));

  try {
    const payload = tokenUtils.validateAccessToken(token);

    if (payload.type !== TokenType.AUTH) {
      throw new BadRequestError("Token inválido");
    }
  }
```

```

    req.user = {
      id: payload.id,
      role: payload.role,
    };
    next();
  } catch (err: any) {
    if (err.name === "TokenExpiredError") {
      return next(new ForbiddenError("Token expirado"));
    }
    next(new ForbiddenError("Token inválido"));
  }
}

function getBearerToken(req: Request): string | null {
  const header = req.headers["authorization"];
  if (!header) return null;

  const parts = header.split(" ");
  if (parts.length !== 2) return null;

  const [scheme, token] = parts;
  if (scheme !== "Bearer") return null;

  return token;
}

```

BruteForceProtection.ts

```

import { Request, Response, NextFunction } from "express";
import { TooManyRequest } from "../utils/Errors/TooManyRequest";
import { plainToInstance } from "class-transformer";
import { UserDTO } from "../dtos/user/UserDto";
import { BruteForceProtectionRepository } from
  "../repositories/BruteForceProtectionRepository";

const bruteRepo = new BruteForceProtectionRepository();

export default async function BruteForceProtection(
  req: Request,
  res: Response,
  next: NextFunction
) {
  const dto = plainToInstance(UserDTO, req.body);
  const username = dto.email;
  if (!username) return next();

  try {

```

```

const rate = await bruteRepo.getAttempts(username);
if (rate && rate.consumedPoints >= bruteRepo.maxAttempts) {
  const retrySecs = Math.round(rate.msBeforeNext / 1000) || 60;
  throw new TooManyRequest(
    `Conta temporariamente bloqueada. Tente novamente em ${retrySecs} segundos.`
  );
}
next();
} catch (err) {
  next(err);
}
}

```

Error.ts

```

import { Request, Response, NextFunction } from "express";
import { ErrorResponse } from "../utils/responses/ErrorResponse";
import Logger from "../config/Logger";
import { UniqueConstraintError, ValidationError } from "sequelize";
import { BadRequestError } from "../utils/errors/BadRequestError";
const errorHandlers = [
  {
    type: SyntaxError,
    status: 400,
    message: "JSON inválido no corpo da requisição.",
  },
  { type: UniqueConstraintError, status: 409, message: "Registro duplicado." },
  { type: ValidationError, status: 422, message: "Dados inválidos." },
];

```

```

export function ErrorMiddleware(
  err: any,
  req: Request,
  res: Response,
  next: NextFunction
) {
  for (const handler of errorHandlers) {
    if (err instanceof handler.type) {
      return res
        .status(handler.status)
        .json(ErrorResponse(handler.message, handler.status));
    }
  }
}

```

```

if (err instanceof BadRequestError) {
  return res.status(err.statusCode).json({
    success: false,
    message: err.message,
    statusCode: err.statusCode,
    errors: err.errors?.map((e) => ({ field: e.field, message: e.message })),
  });
}

if (err.statusCode) {
  return res
    .status(err.statusCode)
    .json(ErrorResponse(err.message, err.statusCode));
}

Logger.error("Erro:", err);
if (err instanceof Error) {
  Logger.error(`Stack: ${err.stack}`);
}

return res.status(500).json(ErrorResponse("Erro interno do servidor.", 500));
}

```

GatewayMiddleware.ts

```

import { Socket } from "socket.io";
import Logger from "../config/Logger";
import { TokenType } from "../enums	TokenName/TokenType";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { TokenUtils } from "../utils	TokenName/TokenUtils";
import { ForbiddenError } from "../utils/errors/ForbiddenError";

var tokenUtils = new TokenUtils();

export function GatewayMiddleware(socket: Socket, next: any) {
  socket.data.user = null;

  try {
    const token = extractTokenFromSocket(socket);

    if (!token) {
      Logger.warn("Tentativa de conexão não autorizada", {
        ip: socket.handshake.address,
        userAgent: socket.handshake.headers["user-agent"],
      });
    }
  }
}

```

```

    });

    return next(new ForbiddenError("Token de autenticação necessário"));
}

const payload = validateToken(token);

socket.data.user = {
  id: payload.id,
  role: payload.role,
  email: payload.email,
  name: payload.name,
};

Logger.info(`Socket autenticado: Usuário ${payload.id} (${payload.role})`);

next();
} catch (error: any) {
  handleAuthError(error, next);
}
}

function extractTokenFromSocket(socket: Socket): string | null {
  const headerToken =
    socket.handshake.headers.authorization?.replace("Bearer ", "") || null;

  if (headerToken) {
    return headerToken;
  }

  const authToken =
    socket.handshake.auth.authorization?.replace("Bearer ", "") || null;

  return authToken;
}

function validateToken(token: string) {
  const payload = tokenUtils.validateAccessToken(token);

  if (payload.type !== TokenType.AUTH) {
    throw new BadRequestError("Tipo de token inválido para autenticação");
  }

  return payload;
}

```

```

function handleAuthError(error: any, next: any) {
  Logger.error("Erro na autenticação do socket:", {
    error: error.message,
    type: error.name,
  });

  if (error.name === "TokenExpiredError") {
    return next(new ForbiddenError("Sessão expirada. Faça login novamente."));
  }

  if (error.name === "JsonWebTokenError") {
    return next(new ForbiddenError("Token de autenticação inválido."));
  }

  if (error instanceof BadRequestError || error instanceof ForbiddenError) {
    return next(error);
  }

  return next(new ForbiddenError("Falha na autenticação"));
}

```

JsonRequired.ts

```

import { Request, Response, NextFunction } from "express";
import { ErrorResponse } from "../utils/responses/ErrorResponse";

export default function JsonRequiredMiddleware(
  req: Request,
  res: Response,
  next: NextFunction
): void {

  if (req.headers["content-type"] !== "application/json") {
    res
      .status(415)
      .json(ErrorResponse("A requisição deve ser no formato JSON", 415));
    return;
  }

  next();
}

```

Limiter.ts

```
import rateLimit from "express-rate-limit";
import { Request, Response, NextFunction } from "express";
import { TooManyRequest } from "../utils/Errors/TooManyRequest";

export const Limiter = rateLimit({
  windowMs: 15 * 60 * 60, // 15 minutos
  max: 7,
  message: "Muitas requisições! Tente novamente mais tarde.",
  standardHeaders: true,
  legacyHeaders: false,
  keyGenerator: (req: Request): string => {
    const ip = req.headers["x-forwarded-for"];
    if (typeof ip === "string") return ip;
    if (Array.isArray(ip)) return ip[0];
    return req.socket.remoteAddress || "unknown";
  },
  handler: (req: Request, res: Response, next: NextFunction) => {
    throw new TooManyRequest("Muitas requisições. Tente novamente mais tarde.");
  },
});
```

LoggerMiddleware.ts

```
import { NextFunction, Request, Response } from "express";
import { ILoggerOptions } from "../interfaces/logger/ILoggerOptions";
import Logger from "../config/Logger";
import { LogService } from "../service/LogService";

export function LoggerMiddleware(
  options: ILoggerOptions
): (req: Request, res: Response, next: NextFunction) => void {
  return (req: Request, res: Response, next: NextFunction) => {
    res.on("finish", async () => {
      try {
        if (res.statusCode < 400) {
          const logService = new LogService();
          const actionWithMetadata = `${options.action} - ${req.method} ${req.path}`;
          await logService.createLog({
            userId: (req as any).user?.id || null,
            action: actionWithMetadata,
            module: options.resource,
          });
        }
      } catch (error) {
        console.error(`Error creating log: ${error}`);
      }
    });
  };
}
```

```

        originIp: req.ip,
        userAgent: req.get("User-Agent") || "",
    });
}
} catch (error) {
    Logger.error("Erro ao registrar log:", error);
}
});
};

next();
};
}

```

Maintenance.ts

```

import { Request, Response, NextFunction } from "express";
import { MaintenanceResponse } from "../utils/responses/MaintenanceResponse";

export default function MaintenanceMiddleware(
    req: Request,
    res: Response,
    next: NextFunction
): void {
    const isMaintenance = process.env.MAINTENANCE === "true";

    if (isMaintenance) {
        res.status(503).json(MaintenanceResponse());
        return;
    }

    next();
}

```

NotFound.ts

```

import { Request, Response, NextFunction } from "express";
import { ErrorResponse } from "../utils/responses/ErrorResponse";

export function NotFoundMiddleware(
    req: Request,
    res: Response,
    next: NextFunction
) {

```

```

const responseBody = ErrorResponse("Rota não encontrada", 404);
res.status(404).json(responseBody);
}

```

RegisterEmployeeMiddleware.ts

```

import { Request, Response, NextFunction } from "express";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { ForbiddenError } from "../utils/errors/ForbiddenError";
import { TokenUtils } from "../utils/TokenUtils";
import { TokenType } from "../enums/Token/TokenType";

var tokenUtils = new TokenUtils();

export async function RegisterEmployeeMiddleware(
  req: Request,
  res: Response,
  next: NextFunction
) {
  const token = getBearerToken(req);
  if (!token) return next(new BadRequestError("Token ausente"));

  try {
    const payload = tokenUtils.validateEmployeeToken(token);

    if (payload.type !== TokenType.GENERATE_ACCOUNT) {
      throw new BadRequestError("Token inválido");
    }

    req.user = { id: payload.id };

    next();
  } catch (err) {
    if (err.name === "TokenExpiredError") {
      return next(new ForbiddenError("Token expirado"));
    }
    next(new ForbiddenError("Token inválido"));
  }
}

function getBearerToken(req: Request): string | null {
  const header = req.headers["authorization"];
  if (!header) return null;
}

```

```

const parts = header.split(" ");
if (parts.length !== 2) return null;

const [scheme, token] = parts;
if (scheme !== "Bearer") return null;

return token;
}

```

ResetEmailMiddleware.ts

```

import { Request, Response, NextFunction } from "express";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { ForbiddenError } from "../utils/errors/ForbiddenError";
import { TokenUtils } from "../utils/TokenUtils";
import { TokenType } from "../enums/Token/TokenType";

var tokenUtils = new TokenUtils();

export async function ResetEmailMiddleware(
  req: Request,
  res: Response,
  next: NextFunction
) {
  const token = getBearerToken(req);
  if (!token) return next(new BadRequestError("Token ausente"));

  try {
    const payload = tokenUtils.validateEmailToken(token);

    if (payload.type !== TokenType.RESET_EMAIL) {
      throw new BadRequestError("Token inválido");
    }

    req.user = { id: payload.id, email: payload.email };

    next();
  } catch (err: any) {
    if (err.name === "TokenExpiredError") {
      return next(new ForbiddenError("Token expirado"));
    }
    next(new ForbiddenError("Token inválido"));
  }
}

```

```

function getBearerToken(req: Request): string | null {
  const header = req.headers["authorization"];
  if (!header) return null;

  const parts = header.split(" ");
  if (parts.length !== 2) return null;

  const [scheme, token] = parts;
  if (scheme !== "Bearer") return null;

  return token;
}

```

ResetPassMiddleware.ts

```

import { Request, Response, NextFunction } from "express";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { ForbiddenError } from "../utils/errors/ForbiddenError";
import { TokenUtils } from "../utils/TokenUtils";
import { TokenType } from "../enums/Token/TokenType";

var tokenUtils = new TokenUtils();

export async function ResetPassMiddleware(
  req: Request,
  res: Response,
  next: NextFunction
) {
  const token = getBearerToken(req);
  if (!token) return next(new BadRequestError("Token ausente"));

  try {
    const payload = tokenUtils.validateResetPassToken(token);

    if (payload.type !== TokenType.RESET_PASS) {
      throw new BadRequestError("Token inválido");
    }

    req.user = { id: payload.id };

    next();
  } catch (err: any) {
    if (err.name === "TokenExpiredError") {

```

```

        return next(new ForbiddenError("Token expirado"));
    }
    next(new ForbiddenError("Token inválido"));
}
}

function getBearerToken(req: Request): string | null {
    const header = req.headers["authorization"];
    if (!header) return null;

    const parts = header.split(" ");
    if (parts.length !== 2) return null;

    const [scheme, token] = parts;
    if (scheme !== "Bearer") return null;

    return token;
}

```

ValidateRequest.ts

```

import { validate } from "class-validator";
import { Request, Response, NextFunction } from "express";
import { plainToInstance } from "class-transformer";
import { ErrorResponse } from "../utils/responses/ErrorResponse";

export function ValidateRequest<T extends object>(
    dtoClass: new () => T,
    groups?: string[],
    source: "body" | "query" | "params" = "body"
) {
    return async (req: Request, res: Response, next: NextFunction) => {
        const data = req[source];
        const dtoInstance = plainToInstance(dtoClass, data);

        const errors = await validate(dtoInstance, { groups });

        if (errors.length > 0) {
            const errorDetails = errors.flatMap((error) =>
                Object.entries(error.constraints || {}).map(([_, message]) => ({
                    field: error.property,
                    message,
                }))
            );
        }
    };
}

```

```

    res
      .status(400)
      .json(ErrorResponse("Requisição inválida", 400, errorDetails));
    return;
  }

  next();
};

}

```

ValidateRoles.ts

```

import { Request, Response, NextFunction } from "express";
import { UserRole } from "../enums/User/UserRole";
import { BadRequestError } from "../utils/errors/BadRequestError";

export function ValidateRoles(neededRole: UserRole) {
  return async (req: Request, _res: Response, next: NextFunction) => {
    try {
      const loggedInUser = req.user;

      if (!loggedInUser) {
        return next(new BadRequestError("Usuário não autenticado"));
      }

      if (loggedInUser.role === undefined || loggedInUser.role === null) {
        return next(new BadRequestError("Usuário sem função definida"));
      }

      const rolesHierarchy = {
        [UserRole.PATIENT]: 0,
        [UserRole.EMPLOYEE]: 1,
        [UserRole.ADMIN]: 2,
      };

      const userRoleLevel = rolesHierarchy[loggedInUser.role];
      const neededRoleLevel = rolesHierarchy[neededRole];

      if (userRoleLevel < neededRoleLevel) {
        return next(new BadRequestError("Acesso negado"));
      }

      next();
    }
  }
}

```

```
    } catch (err) {  
        next(err);  
    }  
};  
}
```

AttendanceRepository.ts

```
import { Op, Sequelize } from "sequelize";
import Attendance from "../database/models/Attendance";
import { ICreateAttendanceInput } from "../interfaces/attendance/ICreateAttendanceInput";
import User from "../database/models/User";
import { IUpdateAttendanceInput } from "../interfaces/attendance/IUpdateAttendanceInput";

export class AttendanceRepository {
  async create(data: ICreateAttendanceInput, options?: any) {
    return Attendance.create(
      {
        userId: data.userId,
        patientId: data.patientId,
        entryDate: data.entryDate,
        attendanceDate: data.attendanceDate,
        dischargeDate: data.dischargeDate,
        priority: data.priority,
        mainComplaint: data.mainComplaint,
        currentIllnessHistory: data.currentIllnessHistory,
        allergies: data.allergies,
        currentMedications: data.currentMedications,
        mainDiagnosis: data.mainDiagnosis,
        secondaryDiagnoses: data.secondaryDiagnoses,
        proceduresPerformed: data.proceduresPerformed,
        destination: data.destination,
        referralTo: data.referralTo,
        dischargeNotes: data.dischargeNotes,
        status: data.status,
      },
      options
    );
  }

  async findById(id: number, options?: any) {
    const attendance = await Attendance.findByPk(id, options);
    return attendance;
  }
}
```

```

async updateById(
  id: number,
  data: IUpdateAttendanceInput,
  options?: any
): Promise<Attendance | null> {
  await this.findById(id);
  const [affectedCount] = await Attendance.update(data, {
    where: { id },
    ...options,
  });
  return await this.findById(id, options);
}

async deleteById(id: number, options?: any) {
  await this.findById(id);
  return Attendance.destroy({
    where: { id },
    ...options,
  });
}

async findPaginated(params: {
  offset: number;
  limit: number;
  patientId?: number;
  startDate?: Date;
  endDate?: Date;
}) {
  const { offset, limit, patientId, startDate, endDate } = params;
  const whereClause: any = {};
  if (patientId) {
    whereClause.patientId = {
      [Op.eq]: patientId,
    };
  }
  if (startDate || endDate) {
    whereClause.entryDate = {};
    if (startDate) {
      whereClause.entryDate[Op.gte] = startDate;
    }
  }
}

```

```

if (endDate) {
  whereClause.entryDate[Op.lte] = endDate;
}

const { count, rows } = await Attendance.findAndCountAll({
  where: whereClause,
  limit,
  offset,
  order: [
    ["entryDate", "DESC"],
    [Sequelize.literal(`data_criacao`), "DESC"],
  ],
  include: [
    {
      association: "patient",
      attributes: ["id", "cpf"],
      include: [
        {
          model: User,
          as: "user",
          attributes: ["name"],
        },
      ],
    },
  ],
});
}

return {
  attendances: rows,
  total: count,
};
}
}

```

BruteForceProtectionRepository.ts

```

import { getRedis } from "../config/Redis";
import { RateLimiterRedis } from "rate-limiter-flexible";
import Redis from "ioredis";

export class BruteForceProtectionRepository {
  private redis: Redis;

```

```

maxAttempts = 5;
limiterDuration = 30 * 60;

private getLimiter(): RateLimiterRedis {
    if (!this.redis) {
        this.redis = getRedis();
    }

    return new RateLimiterRedis({
        storeClient: this.redis,
        keyPrefix: "login_fail",
        points: this.maxAttempts,
        duration: this.limiterDuration,
    });
}

async registerFailedAttempt(username: string) {
    await this.getLimiter().consume(username.toLowerCase());
}

async getAttempts(username: string) {
    return await this.getLimiter().get(username);
}

async resetAttempts(username: string) {
    await this.getLimiter().delete(username.toLowerCase());
}
}

```

LogRepository.ts

```

import { Op } from "sequelize";
import Log from "../database/models/Log";
import { ILogFilter } from "../interfaces/log/ILogFilter";
import { ILogCreate } from "../interfaces/log/ILogCreate";

export class LogRepository {
    async create(data: ILogCreate) {
        return Log.create({
            userId: data.userId,
            action: data.action,
            module: data.module,
            originIp: data.originIp,
        })
    }
}

```

```

    userAgent: data.userAgent,
  });
}

async findById(id: number) {
  return Log.findOne({ where: { id } });
}

async findByUserId(userId: number) {
  return Log.findAll({
    where: { userId },
    order: [["createdAt", "DESC"]],
  });
}

async findPaginated(params: ILogFilter) {
  const { offset, limit, userId, action, module } = params;
  const whereClause: any = {};

  if (userId) {
    whereClause.userId = {
      [Op.eq]: userId,
    };
  }

  if (action) {
    whereClause.action = {
      [Op.like]: `%${action}%`,
    };
  }

  if (module) {
    whereClause.module = {
      [Op.like]: `%${module}%`,
    };
  }

  const { count, rows } = await Log.findAndCountAll({
    where: whereClause,
    limit,
    offset,
    order: [["createdAt", "DESC"]],
  });
}

```

```

        return {
          logs: rows,
          total: count,
        };
      }
    }
  }
}

```

ManagerRepository.ts

```

import Manager from "../database/models/Manager";
import { ICreateManagerInput } from "../interfaces/manager/ICreateManagerInput";

export class ManagerRepository {
  async create(data: ICreateManagerInput) {
    return Manager.create({
      userId: data.userId,
      hospitalCode: data.hospitalCode,
    });
  }

  async findById(id: number) {
    return Manager.findByPk(id);
  }

  async findByHospitalCode(hospitalCode: string) {
    return Manager.findOne({
      where: { hospitalCode },
    });
  }

  async findByUserId(userId: number) {
    return Manager.findOne({ where: { userId } });
  }
}

```

PatientRepository.ts

```

import { Patient } from "../database/models/Patient";
import User from "../database/models/User";
import { PatientDto } from "../dtos/patient/PatientDto";
import { BadRequestError } from "../utils/errors/BadRequestError";

export class PatientRepository {

```

```
async findById(id: number) {
  return Patient.findOne({
    where: { id },
    include: [
      {
        model: User,
        as: "user",
        attributes: ["name"],
      },
    ],
  });
}

async findByUserId(userId: number) {
  return Patient.findOne({
    where: { userId },
    include: [
      {
        model: User,
        as: "user",
        attributes: ["name", "email"],
      },
    ],
  });
}

async create(data: Partial<PatientDto> & { userId: number }) {
  const patient = await Patient.create(data);
  return await this.findByUserId(patient.userId);
}

async update(id: number, data: Partial<PatientDto>) {
  const patient = await this.findById(id);
  if (!patient) throw new BadRequestError("Paciente não encontrado.");
  await patient.update(data);
  return patient;
}

async delete(id: number): Promise<void> {
  const patient = await this.findById(id);
  if (!patient) throw new BadRequestError("Paciente não encontrado.");

  await patient.destroy();
}
```

```

async updateById(
  userId: number,
  data: Partial<PatientDto>
): Promise<Patient> {
  const patient = await this.findById(userId);
  if (!patient) throw new BadRequestError("Paciente não encontrado.");

  const sanitizedData = Object.fromEntries(
    Object.entries(data).filter(([_, value]) => {
      return value !== undefined && value !== null && value !== "";
    })
  );

  await patient.update(sanitizedData);
  return patient;
}

```

QueueEventRepository.ts

```

import QueueEvent from "../database/models/QueueEvent";
import { QueueType } from "../enums/Queue/QueueType";
import { QueueTypePT } from "../enums/Queue/QueueTypePT";
import {
  IAvgWait,
  ICreateQueueEventInput,
} from "../interfaces/queue/ICreateQueueEventInput";
import { Op, QueryTypes } from "sequelize";

export class QueueEventRepository {
  async create(queue: QueueType, data: ICreateQueueEventInput) {
    const queueName = QueueTypePT[queue as QueueType];

    return await QueueEvent.create({
      queueId: data.queueId,
      queue: queueName.toLowerCase(),
      patientId: data.patientId,
      enteredAt: data.enteredAt,
      startedAt: data.startedAt,
    });
  }

  async getAverageWaitTimeByQueue(

```

```

queue: QueueType,
startDate?: Date,
endDate?: Date
) {
  const whereClause: any = {
    queue: QueueTypePT[queue],
    startedAt: { [Op.not]: null },
  };

  if (startDate && endDate) {
    whereClause.enteredAt = { [Op.between]: [startDate, endDate] };
  }

  const result = (await QueueEvent.findOne({
    where: whereClause,
    attributes: [
      [
        QueueEvent.sequelize!.literal(
          `ROUND(AVG(TIMESTAMPDIFF(SECOND, entrou_em, inicio_em)))``,
        ),
        "avgWaitSeconds",
      ],
      ],
    raw: true,
  })) as IAvgWait | null;

  return result?.avgWaitSeconds ?? 0;
}

async findById(id: string) {
  return QueueEvent.findByPk(id);
}

async findByQueueId(queueId: string) {
  return QueueEvent.findOne({ where: { queueId } });
}

async findByPatientId(patientId: string) {
  return QueueEvent.findAll({ where: { patientId } });
}

async findByQueue(queue: "triagem" | "atendimento") {
  return QueueEvent.findAll({ where: { queue } });
}

```

```

async findByDateRange(startDate: Date, endDate: Date) {
  return QueueEvent.findAll({
    where: {
      enteredAt: {
        [Op.between]: [startDate, endDate],
      },
    },
    order: [["entrou_em", "ASC"]],
  });
}

async getPeakDemandByWeek(startDate: Date, endDate: Date) {
  const result = await QueueEvent.sequelize!.query(
    `

      SELECT
        DATE_FORMAT(entrou_em, '%a') as day,
        COUNT(*) as total_entries
      FROM fila_eventos
      WHERE entrou_em BETWEEN ? AND ?
      GROUP BY DATE_FORMAT(entrou_em, '%a')
    `,
    {
      replacements: [startDate, endDate],
      type: QueryTypes.SELECT,
    }
  );
}

const dayMap: Record<string, string> = {
  Sun: "DOM",
  Mon: "SEG",
  Tue: "TER",
  Wed: "QUA",
  Thu: "QUI",
  Fri: "SEX",
  Sat: "SAB",
};

return (result as any[]).reduce(
  (acc, row) => ({
    ...acc,
    [dayMap[row.day] || row.day]: Number(row.total_entries),
  }),
  {}
)

```

```

    );
}

async getMostFrequentWaitTimes(startDate: Date, endDate: Date) {
  const result = await QueueEvent.sequelize!.query(
    `

    WITH wait_times AS (
      SELECT
        LOWER(TRIM(fila)) AS queue,
        ROUND(TIMESTAMPDIFF(SECOND, entrou_em, iniciou_em)) AS
        wait_time_seconds,
        COUNT(*) AS frequency
      FROM fila_eventos
      WHERE entrou_em BETWEEN ? AND ?
        AND iniciou_em IS NOT NULL
        AND entrou_em IS NOT NULL
        AND TIMESTAMPDIFF(SECOND, entrou_em, iniciou_em) > 0
      GROUP BY LOWER(TRIM(fila)), ROUND(TIMESTAMPDIFF(SECOND, entrou_em,
        iniciou_em)))
    ),
    ranked_times AS (
      SELECT
        queue,
        wait_time_seconds,
        frequency,
        ROW_NUMBER() OVER (
          PARTITION BY
            CASE
              WHEN queue LIKE '%atendimento%' THEN 'atendimento'
              WHEN queue LIKE '%triagem%' THEN 'triagem'
            END
          ORDER BY frequency DESC
        ) as row_num
      FROM wait_times
      WHERE queue LIKE '%atendimento%' OR queue LIKE '%triagem%'
    )
    SELECT queue, wait_time_seconds, frequency
    FROM ranked_times
    WHERE row_num <= 5
    ORDER BY
      CASE
        WHEN queue LIKE '%atendimento%' THEN 1
        WHEN queue LIKE '%triagem%' THEN 2
      END,
  `
  );
}

```

```

frequency DESC
`,
{
  replacements: [startDate, endDate],
  type: QueryTypes.SELECT,
}
);

console.log("Resultado filtrado:", result);

const grouped = (result as any[]).reduce(
  (acc, row) => {
    const item = {
      averageWaitTime: Number(row.wait_time_seconds),
      count: Number(row.frequency),
    };

    if (row.queue.includes("atendimento")) {
      acc.atendimento.push(item);
    } else if (row.queue.includes("triagem")) {
      acc.triagem.push(item);
    }
  }

  return acc;
},
{
  atendimento: [] as { averageWaitTime: number; count: number }[],
  triagem: [] as { averageWaitTime: number; count: number }[],
}
);

return grouped;
}

async getPatientJourney(patientId: string) {
  return QueueEvent.findAll({
    where: { patientId },
    order: [["entrou_em", "ASC"]],
  });
}

async update(id: string, data: Partial<ICreateQueueEventInput>) {
  const record = await QueueEvent.findByPk(id);
  return record?.update(data) || null;
}

```

```

    }

    async delete(id: string) {
        return QueueEvent.destroy({ where: { id } });
    }
}

export default QueueEventRepository;

```

QueueRepository.ts

```

import { RedisClientType } from "redis";
import { QueueStatus } from "../enums/Queue/QueueStatus";
import { getRedis } from "../config/Redis";
import Redis from "ioredis";
import { IQueueHistoryResponse } from "../interfaces/queue/IQueueHistoryResponse";
import { QueueType } from "../enums/Queue/QueueType";

export class QueueRepository {
    private redis: Redis;

    private getRedisClient(): Redis {
        if (!this.redis) {
            this.redis = getRedis();
        }
        return this.redis;
    }

    async getAllPatientsWithQueueData(): Promise<number[]> {
        const redis = this.getRedisClient();
        const keys = await redis.keys("patient:*:queueData");

        return keys
            .map((key) => {
                const match = key.match(/patient:(\d+):queueData/);
                return match ? Number(match[1]) : null;
            })
            .filter((id): id is number => id !== null);
    }

    async getPatientMeta(patientId: number): Promise<Record<string, string>> {
        const redis = this.getRedisClient();
        const meta = await redis.hgetall(`patient:${patientId}:queueData`);
        return meta || {};
    }
}

```

```

}

async setPatientMeta(patientId: number, meta: Record<string, string>) {
  const redis = this.getRedisClient();
  await redis.hset(`patient:${patientId}:queueData`, meta);
}

async deletePatientMeta(patientId: number) {
  const redis = this.getRedisClient();
  await redis.del(`patient:${patientId}:queueData`);
}

async addPatientToQueue(
  queueType: QueueType,
  patientId: number,
  score: number
) {
  const redis = this.getRedisClient();
  const queueKey = `queue:${queueType}`;

  const invertedScore = -score;
  await redis.zadd(queueKey, invertedScore, patientId.toString());
}

async removePatientFromHistory(queueType: QueueType, patientId: number) {
  const redis = this.getRedisClient();
  const historyKey = `queue:history:${queueType}`;
  await redis.zrem(historyKey, patientId.toString());
}

async removePatientFromQueue(queueType: QueueType, patientId: number) {
  const redis = this.getRedisClient();
  const queueKey = `queue:${queueType}`;
  await redis.zrem(queueKey, patientId.toString());
}

async addPatientToCalledHistory(
  queueType: QueueType,
  patientId: number,
  name: string,
  room: string
): Promise<void> {
  const timestamp = Date.now();
  const historyKey = `queue:called:history:${queueType}`;
}

```

```

await this.redis
  .multi()
  .hset(`queue:called:${queueType}:${patientId}`, {
    name,
    room,
    timestamp: timestamp.toString(),
  })
  .zadd(historyKey, timestamp, patientId.toString())
  .zremrangebyrank(historyKey, 0, -7)
  .exec();
}

async getLastCalledPatients(
  queueType: QueueType
): Promise<IQueueHistoryResponse[]> {
  const redis = this.getRedisClient();
  const limit = 6;
  const historyKey = `queue:called:history:${queueType}`;

  const patientIds = await redis.zrevrange(historyKey, 0, limit - 1);

  const patients: IQueueHistoryResponse[] = [];

  for (const patientId of patientIds) {
    const data = await redis.hgetall(
      `queue:called:${queueType}:${patientId}`
    );
    if (data && Object.keys(data).length > 0) {
      patients.push({
        name: data.name,
        room: data.room,
        timestamp: Number(data.timestamp),
      });
    }
  }

  return patients;
}

async getQueuePatients(
  queueType: QueueType,
  start: number,
  end: number
): Promise<string[]> {

```

```

const redis = this.getRedisClient();
const queueKey = `queue:${queueType}`;
return await redis.zrange(queueKey, start, end);
}

async getQueueLength(queueType: QueueType): Promise<number> {
  const redis = this.getRedisClient();
  const queueKey = `queue:${queueType}`;
  return await redis.zcard(queueKey);
}

async getNextPatientId(queueType: QueueType): Promise<string | null> {
  const redis = this.getRedisClient();
  const queueKey = `queue:${queueType}`;

  const ids = await redis.zrange(queueKey, 0, 0);
  return ids.length > 0 ? ids[0] : null;
}

async getPatientPosition(queueType: QueueType, patientId: number) {
  const queueKey = `queue:${queueType}`;
  const rank = await this.redis.zrank(queueKey, patientId.toString());
  return rank !== null ? rank + 1 : 0;
}

async getLastTimestamps(queueType: QueueType, limit = 10) {
  const historyKey = `queue:history:${queueType}`;
  const timestamps = await this.redis.zrange(historyKey, -limit, -1);
  return timestamps.map((t) => Number(t));
}

async setPatientStatus(patientId: number, status: QueueStatus) {
  const redis = this.getRedisClient();
  await redis.hset(`patient:${patientId}:queueData`, { status });
}

async registerPatientCalled(queueType: string, patientId: number) {
  const historyKey = `queue:history:${queueType}`;
  const timestamp = Date.now();
  await this.redis.zadd(
    historyKey,
    timestamp.toString(),
    patientId.toString()
  );
}

```

```
}

async patientExistsInQueue(patientId: number): Promise<boolean> {
  const meta = await this.getPatientMeta(patientId);
  return meta && Object.keys(meta).length > 0 && !meta.type;
}

}
```

TokenRepository.ts

```
import { RefreshToken } from "../database/models/RefreshToken";

export class TokenRepository {
    async saveRefreshToken(userId: number, token: string) {
        return RefreshToken.create({ userId, token });
    }

    async findRefreshToken(userId: number, token: string) {
        return RefreshToken.findOne({ where: { userId, token } });
    }

    async findRefreshTokenByUser(userId: number) {
        return RefreshToken.findOne({ where: { userId } });
    }

    async saveOrUpdateRefreshToken(userId: number, newToken: string) {
        const existing = await this.findRefreshTokenByUser(userId);
        if (existing) {
            existing.token = newToken;
            await existing.save();
            return;
        }

        await this.saveRefreshToken(userId, newToken);
    }

    async deleteRefreshToken(userId: number, token: string) {
        return RefreshToken.destroy({ where: { usuario_id: userId, token } });
    }
}
```

UserRepository.ts

```

import { Op } from "sequelize";
import User from "../database/models/User";
import { UserRole } from "../enums/User/UserRole";
import { ICreateUserInput } from "../interfaces/user/ICreateUser";
import { BadRequestError } from "../utils/errors/BadRequestError";

export class UserRepository {
  async create(data: ICreateUserInput, options?: any) {
    return User.create(
      {
        name: data.name,
        email: data.email,
        password: data.password,
        role: data.role,
      },
      options
    );
  }

  async findByEmail(email: string) {
    return User.findOne({ where: { email, deletado: false } });
  }

  async findById(id: number) {
    return User.findOne({ where: { id, deletado: false } });
  }

  async findPaginated(params: {
    offset: number;
    limit: number;
    name?: string;
    role?: UserRole;
  }) {
    const { offset, limit, name, role } = params;
    const whereClause: any = {};

    if (name) {
      whereClause.name = {
        [Op.like]: `%${name}%`,
      };
    }

    if (role !== undefined) {
      whereClause.role = {
        [Op.eq]: role,
      };
    }
  }
}

```

```
const { count, rows } = await User.findAndCountAll({
  where: whereClause,
  limit,
  offset,
  order: [["data_criacao", "DESC"]],
});

return {
  users: rows,
  total: count,
};

}

async delete(id: number) {
  const user = await this.findById(id);
  if (!user) throw new BadRequestError("Usuário não encontrado.");
  user.deleted = true;
  await user.save();
  return user;
}

async updateRole(newRole: UserRole, userId: number) {
  const user = await this.findById(userId);
  if (!user) throw new BadRequestError("Usuário não encontrado.");
  user.role = newRole;
  await user.save();
  return user;
}

async updateName(newName: string, userId: number) {
  const user = await this.findById(userId);
  if (!user) throw new BadRequestError("Usuário não encontrado.");
  user.name = newName;
  await user.save();
  return user;
}

async updatePassword(id: number, hashedPassword: string) {
  const user = await this.findById(id);
  if (!user) throw new BadRequestError("Usuário não encontrado.");
  user.password = hashedPassword;
  await user.save();
  return user;
}

async updateEmail(id: number, email: string) {
  const user = await this.findById(id);
  if (!user) throw new BadRequestError("Usuário não encontrado.");
  user.email = email;
  await user.save();
}
```

```

        return user;
    }
}

```

BruteForceProtection.ts

```

import { Request, Response, NextFunction } from "express";
import { TooManyRequest } from "../utils/Errors/TooManyRequest";
import { plainToInstance } from "class-transformer";
import { UserDTO } from "../dtos/user/UserDto";
import { BruteForceProtectionRepository } from
"../repositories/BruteForceProtectionRepository";

const bruteRepo = new BruteForceProtectionRepository();

export default async function BruteForceProtection(
    req: Request,
    res: Response,
    next: NextFunction
) {
    const dto = plainToInstance(UserDTO, req.body);
    const username = dto.email;
    if (!username) return next();

    try {
        const rate = await bruteRepo.getAttempts(username);
        if (rate && rate.consumedPoints >= bruteRepo.maxAttempts) {
            const retrySecs = Math.round(rate.msBeforeNext / 1000) || 60;
            throw new TooManyRequest(
                `Conta temporariamente bloqueada. Tente novamente em ${retrySecs} segundos.`);
        }
        next();
    } catch (err) {
        next(err);
    }
}

```

Error.ts

```

import { Request, Response, NextFunction } from "express";
import { ErrorResponse } from "../utils/responses/ErrorResponse";
import Logger from "../config/Logger";
import { UniqueConstraintError, ValidationError } from "sequelize";

```

```
import { BadRequestError } from "../utils/errors/BadRequestError";
const errorHandlers = [
  {
    type: SyntaxError,
    status: 400,
    message: "JSON inválido no corpo da requisição.",
  },
  { type: UniqueConstraintError, status: 409, message: "Registro duplicado." },
  { type: ValidationError, status: 422, message: "Dados inválidos." },
];
export function ErrorMiddleware(
  err: any,
  req: Request,
  res: Response,
  next: NextFunction
) {
  for (const handler of errorHandlers) {
    if (err instanceof handler.type) {
      return res
        .status(handler.status)
        .json(ErrorResponse(handler.message, handler.status));
    }
  }

  if (err instanceof BadRequestError) {
    return res.status(err.statusCode).json({
      success: false,
      message: err.message,
      statusCode: err.statusCode,
      errors: err.errors?.map((e) => ({ field: e.field, message: e.message })),
    });
  }

  if (err.statusCode) {
    return res
      .status(err.statusCode)
      .json(ErrorResponse(err.message, err.statusCode));
  }

  Logger.error("Erro:", err);
  if (err instanceof Error) {
    Logger.error(`Stack: ${err.stack}`);
  }
}
```

```

    return res.status(500).json(ErrorResponse("Erro interno do servidor.", 500));
}

```

GatewayMiddleware.ts

```

import { Socket } from "socket.io";
import Logger from "../config/Logger";
import { TokenType } from "../enums/Token/TokenType";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { TokenUtils } from "../utils/TokenUtils";
import { ForbiddenError } from "../utils/errors/ForbiddenError";

var tokenUtils = new TokenUtils();

export function GatewayMiddleware(socket: Socket, next: any) {
  socket.data.user = null;

  try {
    const token = extractTokenFromSocket(socket);

    if (!token) {
      Logger.warn("Tentativa de conexão não autorizada", {
        ip: socket.handshake.address,
        userAgent: socket.handshake.headers["user-agent"],
      });
      return next(new ForbiddenError("Token de autenticação necessário"));
    }

    const payload = validateToken(token);

    socket.data.user = {
      id: payload.id,
      role: payload.role,
      email: payload.email,
      name: payload.name,
    };

    Logger.info(`Socket autenticado: Usuário ${payload.id} (${payload.role})`);

    next();
  } catch (error: any) {
    handleAuthError(error, next);
  }
}

```

```
}

function extractTokenFromSocket(socket: Socket): string | null {
  const headerToken =
    socket.handshake.headers.authorization?.replace("Bearer ", "") || null;

  if (headerToken) {
    return headerToken;
  }

  const authToken =
    socket.handshake.auth.authorization?.replace("Bearer ", "") || null;

  return authToken;
}

function validateToken(token: string) {
  const payload = tokenUtils.validateAccessToken(token);

  if (payload.type !== TokenType.AUTH) {
    throw new BadRequestError("Tipo de token inválido para autenticação");
  }

  return payload;
}

function handleAuthError(error: any, next: any) {
  Logger.error("Erro na autenticação do socket:", {
    error: error.message,
    type: error.name,
  });

  if (error.name === "TokenExpiredError") {
    return next(new ForbiddenError("Sessão expirada. Faça login novamente."));
  }

  if (error.name === "JsonWebTokenError") {
    return next(new ForbiddenError("Token de autenticação inválido."));
  }

  if (error instanceof BadRequestError || error instanceof ForbiddenError) {
    return next(error);
  }

  return next(new ForbiddenError("Falha na autenticação"));
}
```

```
}
```

JsonRequired.ts

```
import { Request, Response, NextFunction } from "express";
import { ErrorResponse } from "../utils/responses/ErrorResponse";

export default function JsonRequiredMiddleware(
  req: Request,
  res: Response,
  next: NextFunction
): void {

  if (req.headers["content-type"] !== "application/json") {
    res
      .status(415)
      .json(ErrorResponse("A requisição deve ser no formato JSON", 415));
    return;
  }

  next();
}
```

Limiter.ts

```
import rateLimit from "express-rate-limit";
import { Request, Response, NextFunction } from "express";
import { TooManyRequest } from "../utils/Errors/TooManyRequest";

export const Limiter = rateLimit({
  windowMs: 15 * 60 * 60, // 15 minutos
  max: 7,
  message: "Muitas requisições! Tente novamente mais tarde.",
  standardHeaders: true,
  legacyHeaders: false,
  keyGenerator: (req: Request): string => {
    const ip = req.headers["x-forwarded-for"];
    if (typeof ip === "string") return ip;
    if (Array.isArray(ip)) return ip[0];
    return req.socket.remoteAddress || "unknown";
  },
  handler: (req: Request, res: Response, next: NextFunction) => {
    throw new TooManyRequest("Muitas requisições. Tente novamente mais tarde.");
  }
});
```

```
  },
});
```

LoggerMiddleware.ts

```
import { NextFunction, Request, Response } from "express";
import { ILoggerOptions } from "../interfaces/logger/ILoggerOptions";
import Logger from "../config/Logger";
import { LogService } from "../service/LogService";

export function LoggerMiddleware(
  options: ILoggerOptions
): (req: Request, res: Response, next: NextFunction) => void {
  return (req: Request, res: Response, next: NextFunction) => {
    res.on("finish", async () => {
      try {
        if (res.statusCode < 400) {
          const logService = new LogService();
          const actionWithMetadata = `${options.action} - ${req.method} ${req.path}`;

          await logService.createLog({
            userId: (req as any).user?.id || null,
            action: actionWithMetadata,
            module: options.resource,
            originIp: req.ip,
            userAgent: req.get("User-Agent") || "",
          });
        }
      } catch (error) {
        Logger.error("Erro ao registrar log:", error);
      }
    });

    next();
  };
}
```

Maintenance.ts

```
import { Request, Response, NextFunction } from "express";
import { MaintenanceResponse } from "../utils/responses/MaintenanceResponse";

export default function MaintenanceMiddleware(
```

```

req: Request,
res: Response,
next: NextFunction
): void {
  const isMaintenance = process.env.MAINTENANCE === "true";

  if (isMaintenance) {
    res.status(503).json(MaintenanceResponse());
    return;
  }

  next();
}

```

NotFound.ts

```

import { Request, Response, NextFunction } from "express";
import { ErrorResponse } from "../utils/responses/ErrorResponse";

export function NotFoundMiddleware(
  req: Request,
  res: Response,
  next: NextFunction
) {
  const responseBody = ErrorResponse("Rota não encontrada", 404);
  res.status(404).json(responseBody);
}

```

RegisterEmployeeMiddleware.ts

```

import { Request, Response, NextFunction } from "express";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { ForbiddenError } from "../utils/errors/ForbiddenError";
import { TokenUtils } from "../utils/TokenUtils";
import { TokenType } from "../enums/Token/TokenType";

var tokenUtils = new TokenUtils();

export async function RegisterEmployeeMiddleware(
  req: Request,
  res: Response,
  next: NextFunction
) {

```

```

const token = getBearerToken(req);
if (!token) return next(new BadRequestError("Token ausente"));

try {
  const payload = tokenUtils.validateEmployeeToken(token);

  if (payload.type !== TokenType.GENERATE_ACCOUNT) {
    throw new BadRequestError("Token inválido");
  }

  req.user = { id: payload.id };

  next();
} catch (err: any) {
  if (err.name === "TokenExpiredError") {
    return next(new ForbiddenError("Token expirado"));
  }
  next(new ForbiddenError("Token inválido"));
}
}

function getBearerToken(req: Request): string | null {
  const header = req.headers["authorization"];
  if (!header) return null;

  const parts = header.split(" ");
  if (parts.length !== 2) return null;

  const [scheme, token] = parts;
  if (scheme !== "Bearer") return null;

  return token;
}

```

ResetEmailMiddleware.ts

```

import { Request, Response, NextFunction } from "express";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { ForbiddenError } from "../utils/errors/ForbiddenError";
import { TokenUtils } from "../utils/TokenUtils";
import { TokenType } from "../enums/Token/TokenType";

var tokenUtils = new TokenUtils();

```

```

export async function ResetEmailMiddleware(
  req: Request,
  res: Response,
  next: NextFunction
) {
  const token = getBearerToken(req);
  if (!token) return next(new BadRequestError("Token ausente"));

  try {
    const payload = tokenUtils.validateEmailToken(token);

    if (payload.type !== TokenType.RESET_EMAIL) {
      throw new BadRequestError("Token inválido");
    }

    req.user = { id: payload.id, email: payload.email };

    next();
  } catch (err: any) {
    if (err.name === "TokenExpiredError") {
      return next(new ForbiddenError("Token expirado"));
    }
    next(new ForbiddenError("Token inválido"));
  }
}

function getBearerToken(req: Request): string | null {
  const header = req.headers["authorization"];
  if (!header) return null;

  const parts = header.split(" ");
  if (parts.length !== 2) return null;

  const [scheme, token] = parts;
  if (scheme !== "Bearer") return null;

  return token;
}

```

ResetPassMiddleware.ts

```

import { Request, Response, NextFunction } from "express";
import { BadRequestError } from "../utils/errors/BadRequestError";
import { ForbiddenError } from "../utils/errors/ForbiddenError";

```

```

import { TokenUtils } from "../utils/TokenUtils";
import { TokenType } from "../enums/Token/TokenType";

var tokenUtils = new TokenUtils();

export async function ResetPassMiddleware(
  req: Request,
  res: Response,
  next: NextFunction
) {
  const token = getBearerToken(req);
  if (!token) return next(new BadRequestError("Token ausente"));

  try {
    const payload = tokenUtils.validateResetPassToken(token);

    if (payload.type !== TokenType.RESET_PASS) {
      throw new BadRequestError("Token inválido");
    }

    req.user = { id: payload.id };

    next();
  } catch (err: any) {
    if (err.name === "TokenExpiredError") {
      return next(new ForbiddenError("Token expirado"));
    }
    next(new ForbiddenError("Token inválido"));
  }
}

function getBearerToken(req: Request): string | null {
  const header = req.headers["authorization"];
  if (!header) return null;

  const parts = header.split(" ");
  if (parts.length !== 2) return null;

  const [scheme, token] = parts;
  if (scheme !== "Bearer") return null;

  return token;
}

```

ValidateRequest.ts

```
import { validate } from "class-validator";
import { Request, Response, NextFunction } from "express";
import { plainToInstance } from "class-transformer";
import { ErrorResponse } from "../utils/responses/ErrorResponse";

export function ValidateRequest<T extends object>(
  dtoClass: new () => T,
  groups?: string[],
  source: "body" | "query" | "params" = "body"
) {
  return async (req: Request, res: Response, next: NextFunction) => {
    const data = req[source];
    const dtoInstance = plainToInstance(dtoClass, data);

    const errors = await validate(dtoInstance, { groups });

    if (errors.length > 0) {
      const errorDetails = errors.flatMap((error) =>
        Object.entries(error.constraints || {}).map(([_, message]) => ({
          field: error.property,
          message,
        }))
      );
      res
        .status(400)
        .json(ErrorResponse("Requisição inválida", 400, errorDetails));
      return;
    }

    next();
  };
}
```

ValidateRoles.ts

```
import { Request, Response, NextFunction } from "express";
import { UserRole } from "../enums/User/UserRole";
import { BadRequestError } from "../utils/errors/BadRequestError";

export function ValidateRoles(neededRole: UserRole) {
  return async (req: Request, _res: Response, next: NextFunction) => {
```

```

try {
  const loggedInUser = req.user;

  if (!loggedInUser) {
    return next(new BadRequestError("Usuário não autenticado"));
  }

  if (loggedInUser.role === undefined || loggedInUser.role === null) {
    return next(new BadRequestError("Usuário sem função definida"));
  }

  const rolesHierarchy = {
    [UserRole.PATIENT]: 0,
    [UserRole.EMPLOYEE]: 1,
    [UserRole.ADMIN]: 2,
  };

  const userRoleLevel = rolesHierarchy[loggedInUser.role];
  const neededRoleLevel = rolesHierarchy[neededRole];

  if (userRoleLevel < neededRoleLevel) {
    return next(new BadRequestError("Acesso negado"));
  }

  next();
} catch (err) {
  next(err);
};

}
}

```

AttendanceRepository.ts

```

import { Op, Sequelize } from "sequelize";
import Attendance from "../database/models/Attendance";
import { ICreateAttendanceInput } from "../interfaces/attendance/ICreateAttendanceInput";
import User from "../database/models/User";
import { IUpdateAttendanceInput } from "../interfaces/attendance/IUpdateAttendanceInput";

export class AttendanceRepository {
  async create(data: ICreateAttendanceInput, options?: any) {
    return Attendance.create(
      {
        userId: data.userId,

```

```

patientId: data.patientId,
entryDate: data.entryDate,
attendanceDate: data.attendanceDate,
dischargeDate: data.dischargeDate,
priority: data.priority,
mainComplaint: data.mainComplaint,
currentIllnessHistory: data.currentIllnessHistory,
allergies: data.allergies,
currentMedications: data.currentMedications,
mainDiagnosis: data.mainDiagnosis,
secondaryDiagnoses: data.secondaryDiagnoses,
proceduresPerformed: data.proceduresPerformed,
destination: data.destination,
referralTo: data.referralTo,
dischargeNotes: data.dischargeNotes,
status: data.status,
},
options
);
}

async findById(id: number, options?: any) {
  const attendance = await Attendance.findById(id, options);
  return attendance;
}

async updateById(
  id: number,
  data: IUpdateAttendanceInput,
  options?: any
): Promise<Attendance | null> {
  await this.findById(id);
  const [affectedCount] = await Attendance.update(data, {
    where: { id },
    ...options,
  });
  return await this.findById(id, options);
}

async deleteById(id: number, options?: any) {
  await this.findById(id);
  return Attendance.destroy({
    where: { id },
    ...options,
  });
}

```

```
    });
}

async findPaginated(params: {
  offset: number;
  limit: number;
  patientId?: number;
  startDate?: Date;
  endDate?: Date;
}) {
  const { offset, limit, patientId, startDate, endDate } = params;
  const whereClause: any = {};

  if (patientId) {
    whereClause.patientId = {
      [Op.eq]: patientId,
    };
  }

  if (startDate || endDate) {
    whereClause.entryDate = {};
    if (startDate) {
      whereClause.entryDate[Op.gte] = startDate;
    }
    if (endDate) {
      whereClause.entryDate[Op.lte] = endDate;
    }
  }

  const { count, rows } = await Attendance.findAndCountAll({
    where: whereClause,
    limit,
    offset,
    order: [
      ["entryDate", "DESC"],
      [Sequelize.literal(`data_criacao`), "DESC"],
    ],
    include: [
      {
        association: "patient",
        attributes: ["id", "cpf"],
        include: [

```

```

    },
    model: User,
    as: "user",
    attributes: ["name"],
),
],
},
],
},
]);
}

return {
    attendances: rows,
    total: count,
};
}
}
}

```

BruteForceProtectionRepository.ts

```

import { getRedis } from "../config/Redis";
import { RateLimiterRedis } from "rate-limiter-flexible";
import Redis from "ioredis";

export class BruteForceProtectionRepository {
    private redis: Redis;

    maxAttempts = 5;
    limiterDuration = 30 * 60;

    private getLimiter(): RateLimiterRedis {
        if (!this.redis) {
            this.redis = getRedis();
        }

        return new RateLimiterRedis({
            storeClient: this.redis,
            keyPrefix: "login_fail",
            points: this.maxAttempts,
            duration: this.limiterDuration,
        });
    }

    async registerFailedAttempt(username: string) {
        await this.getLimiter().consume(username.toLowerCase());
    }
}

```

```

    }

async getAttempts(username: string) {
  return await this.getLimiter().get(username);
}

async resetAttempts(username: string) {
  await this.getLimiter().delete(username.toLowerCase());
}
}

```

LogRepository.ts

```

import { Op } from "sequelize";
import Log from "../database/models/Log";
import { ILogFilter } from "../interfaces/log/ILogFilter";
import { ILogCreate } from "../interfaces/log/ILogCreate";

export class LogRepository {
  async create(data: ILogCreate) {
    return Log.create({
      userId: data.userId,
      action: data.action,
      module: data.module,
      originIp: data.originIp,
      userAgent: data.userAgent,
    });
  }

  async findById(id: number) {
    return Log.findOne({ where: { id } });
  }

  async findByUserId(userId: number) {
    return Log.findAll({
      where: { userId },
      order: [["createdAt", "DESC"]],
    });
  }

  async findPaginated(params: ILogFilter) {
    const { offset, limit, userId, action, module } = params;
    const whereClause: any = {};
  }
}

```

```

if (userId) {
  whereClause.userId = {
    [Op.eq]: userId,
  };
}

if (action) {
  whereClause.action = {
    [Op.like]: `%${action}%`,
  };
}

if (module) {
  whereClause.module = {
    [Op.like]: `%${module}%`,
  };
}

const { count, rows } = await Log.findAndCountAll({
  where: whereClause,
  limit,
  offset,
  order: [["createdAt", "DESC"]],
});
}

return {
  logs: rows,
  total: count,
};
}
}

```

ManagerRepository.ts

```

import Manager from "../database/models/Manager";
import { ICreateManagerInput } from "../interfaces/manager/ICreateManagerInput";

export class ManagerRepository {
  async create(data: ICreateManagerInput) {
    return Manager.create({
      userId: data.userId,
      hospitalCode: data.hospitalCode,
    });
  }
}

```

```

async findById(id: number) {
  return Manager.findByPk(id);
}

async findByHospitalCode(hospitalCode: string) {
  return Manager.findOne({
    where: { hospitalCode },
  });
}

async findByUserId(userId: number) {
  return Manager.findOne({ where: { userId } });
}

```

PatientRepository.ts

```

import { Patient } from "../database/models/Patient";
import User from "../database/models/User";
import { PatientDto } from "../dtos/patient/PatientDto";
import { BadRequestError } from "../utils/errors/BadRequestError";

export class PatientRepository {
  async findById(id: number) {
    return Patient.findOne({
      where: { id },
      include: [
        {
          model: User,
          as: "user",
          attributes: ["name"],
        },
      ],
    });
  }

  async findByUserId(userId: number) {
    return Patient.findOne({
      where: { userId },
      include: [
        {
          model: User,
          as: "user",
        },
      ],
    });
  }
}

```

```
        attributes: ["name", "email"],  
    },  
],  
});  
}  
  
async create(data: Partial<PatientDto> & { userId: number }) {  
    const patient = await Patient.create(data);  
    return await this.findById(patient.userId);  
}  
  
async update(id: number, data: Partial<PatientDto>) {  
    const patient = await this.findById(id);  
    if (!patient) throw new BadRequestError("Paciente não encontrado.");  
    await patient.update(data);  
    return patient;  
}  
  
async delete(id: number): Promise<void> {  
    const patient = await this.findById(id);  
    if (!patient) throw new BadRequestError("Paciente não encontrado.");  
  
    await patient.destroy();  
}  
  
async updateByUserId(  
    userId: number,  
    data: Partial<PatientDto>  
): Promise<Patient> {  
    const patient = await this.findById(userId);  
    if (!patient) throw new BadRequestError("Paciente não encontrado.");  
  
    const sanitizedData = Object.fromEntries(  
        Object.entries(data).filter(([_, value]) => {  
            return value !== undefined && value !== null && value !== "";  
        })  
    );  
  
    await patient.update(sanitizedData);  
    return patient;  
}  
}
```

QueueEventRepository.ts

```

import QueueEvent from "../database/models/QueueEvent";
import { QueueType } from "../enums/Queue/QueueType";
import { QueueTypePT } from "../enums/Queue/QueueTypePT";
import {
  IAvgWait,
  ICreateQueueEventInput,
} from "../interfaces/queue/ICreateQueueEventInput";
import { Op, QueryTypes } from "sequelize";

export class QueueEventRepository {
  async create(queue: QueueType, data: ICreateQueueEventInput) {
    const queueName = QueueTypePT[queue as QueueType];

    return await QueueEvent.create({
      queueId: data.queueId,
      queue: queueName.toLowerCase(),
      patientId: data.patientId,
      enteredAt: data.enteredAt,
      startedAt: data.startedAt,
    });
  }

  async getAverageWaitTimeByQueue(
    queue: QueueType,
    startDate?: Date,
    endDate?: Date
  ) {
    const whereClause: any = {
      queue: QueueTypePT[queue],
      startedAt: { [Op.not]: null },
    };

    if (startDate && endDate) {
      whereClause.enteredAt = { [Op.between]: [startDate, endDate] };
    }

    const result = (await QueueEvent.findOne({
      where: whereClause,
      attributes: [
        [
          QueueEvent.sequelize!.literal(
            `ROUND(AVG(TIMESTAMPDIFF(SECOND, entrhou_em, iniciou_em)))`)),
      ],
    }));
  }
}

```

```

),
"avgWaitSeconds",
],
],
raw: true,
})) as IAvgWait | null;

return result?.avgWaitSeconds ?? 0;
}

async findById(id: string) {
  return QueueEvent.findById(id);
}

async findByQueueId(queueId: string) {
  return QueueEvent.findOne({ where: { queueId } });
}

async findByPatientId(patientId: string) {
  return QueueEvent.findAll({ where: { patientId } });
}

async findByQueue(queue: "triagem" | "atendimento") {
  return QueueEvent.findAll({ where: { queue } });
}

async findByDateRange(startDate: Date, endDate: Date) {
  return QueueEvent.findAll({
    where: {
      enteredAt: {
        [Op.between]: [startDate, endDate],
      },
    },
    order: [["entrou_em", "ASC"]],
  });
}

async getPeakDemandByWeek(startDate: Date, endDate: Date) {
  const result = await QueueEvent.sequelize!.query(
    `SELECT
      DATE_FORMAT(entrou_em, '%a') as day,
      COUNT(*) as total_entries
    FROM fila_eventos
    WHERE entrou_em BETWEEN ? AND ?`,
    {
      replacements: [startDate, endDate],
      type: QueryTypes.SELECT,
    }
  );
  return result;
}

```

```

        WHERE entrou_em BETWEEN ? AND ?
        GROUP BY DATE_FORMAT(entrou_em, '%a')
        ,
        {
            replacements: [startDate, endDate],
            type: QueryTypes.SELECT,
        }
    );
}

const dayMap: Record<string, string> = {
    Sun: "DOM",
    Mon: "SEG",
    Tue: "TER",
    Wed: "QUA",
    Thu: "QUI",
    Fri: "SEX",
    Sat: "SAB",
};

return (result as any[]).reduce(
    (acc, row) => ({
        ...acc,
        [dayMap[row.day] || row.day]: Number(row.total_entries),
    }),
    {}
);
}

async getMostFrequentWaitTimes(startDate: Date, endDate: Date) {
    const result = await QueueEvent.sequelize!.query(
        `

WITH wait_times AS (
    SELECT
        LOWER(TRIM(fila)) AS queue,
        ROUND(TIMESTAMPDIFF(SECOND, entrou_em, iniciou_em)) AS
        wait_time_seconds,
        COUNT(*) AS frequency
    FROM fila_eventos
    WHERE entrou_em BETWEEN ? AND ?
        AND iniciou_em IS NOT NULL
        AND entrou_em IS NOT NULL
        AND TIMESTAMPDIFF(SECOND, entrou_em, iniciou_em) > 0
    GROUP BY LOWER(TRIM(fila)), ROUND(TIMESTAMPDIFF(SECOND, entrou_em,
        iniciou_em))
)
`));
}

```

```

),
ranked_times AS (
  SELECT
    queue,
    wait_time_seconds,
    frequency,
    ROW_NUMBER() OVER (
      PARTITION BY
        CASE
          WHEN queue LIKE '%atendimento%' THEN 'atendimento'
          WHEN queue LIKE '%triagem%' THEN 'triagem'
        END
      ORDER BY frequency DESC
    ) as row_num
  FROM wait_times
  WHERE queue LIKE '%atendimento%' OR queue LIKE '%triagem%'
)
SELECT queue, wait_time_seconds, frequency
FROM ranked_times
WHERE row_num <= 5
ORDER BY
CASE
  WHEN queue LIKE '%atendimento%' THEN 1
  WHEN queue LIKE '%triagem%' THEN 2
END,
frequency DESC
;
{
  replacements: [startDate, endDate],
  type: QueryTypes.SELECT,
}
);

console.log("Resultado filtrado:", result);

const grouped = (result as any[]).reduce(
  (acc, row) => {
    const item = {
      averageWaitTime: Number(row.wait_time_seconds),
      count: Number(row.frequency),
    };
    if (row.queue.includes("atendimento")) {
      acc.atendimento.push(item);
    } else if (row.queue.includes("triagem")) {
      acc.triagem.push(item);
    }
  },
  {
    atendimento: [],
    triagem: []
  }
);

```

```

    } else if (row.queue.includes("triagem")) {
      acc.triagem.push(item);
    }

    return acc;
  },
{
  atendimento: [] as { averageWaitTime: number; count: number }[], 
  triagem: [] as { averageWaitTime: number; count: number }[], 
}
);

return grouped;
}

async getPatientJourney(patientId: string) {
  return QueueEvent.findAll({
    where: { patientId },
    order: [["entrou_em", "ASC"]],
  });
}

async update(id: string, data: Partial<ICreateQueueEventInput>) {
  const record = await QueueEvent.findByPk(id);
  return record?.update(data) || null;
}

async delete(id: string) {
  return QueueEvent.destroy({ where: { id } });
}

export default QueueEventRepository;

```

QueueRepository.ts

```

import { RedisClientType } from "redis";
import { QueueStatus } from "../enums/Queue/QueueStatus";
import { getRedis } from "../config/Redis";
import Redis from "ioredis";
import { IQueueHistoryResponse } from "../interfaces/queue/IQueueHistoryResponse";
import { QueueType } from "../enums/Queue/QueueType";

export class QueueRepository {

```

```

private redis: Redis;

private getRedisClient(): Redis {
  if (!this.redis) {
    this.redis = getRedis();
  }
  return this.redis;
}

async getAllPatientsWithQueueData(): Promise<number[]> {
  const redis = this.getRedisClient();
  const keys = await redis.keys("patient:*:queueData");

  return keys
    .map((key) => {
      const match = key.match(/patient:(\d+):queueData/);
      return match ? Number(match[1]) : null;
    })
    .filter((id): id is number => id !== null);
}

async getPatientMeta(patientId: number): Promise<Record<string, string>> {
  const redis = this.getRedisClient();
  const meta = await redis.hgetall(`patient:${patientId}:queueData`);
  return meta || {};
}

async setPatientMeta(patientId: number, meta: Record<string, string>) {
  const redis = this.getRedisClient();
  await redis.hset(`patient:${patientId}:queueData`, meta);
}

async deletePatientMeta(patientId: number) {
  const redis = this.getRedisClient();
  await redis.del(`patient:${patientId}:queueData`);
}

async addPatientToQueue(
  queueType: QueueType,
  patientId: number,
  score: number
) {
  const redis = this.getRedisClient();
  const queueKey = `queue:${queueType}`;
}

```

```

const invertedScore = -score;
await redis.zadd(queueKey, invertedScore, patientId.toString());
}

async removePatientFromHistory(queueType: QueueType, patientId: number) {
  const redis = this.getRedisClient();
  const historyKey = `queue:history:${queueType}`;
  await redis.zrem(historyKey, patientId.toString());
}

async removePatientFromQueue(queueType: QueueType, patientId: number) {
  const redis = this.getRedisClient();
  const queueKey = `queue:${queueType}`;
  await redis.zrem(queueKey, patientId.toString());
}

async addPatientToCalledHistory(
  queueType: QueueType,
  patientId: number,
  name: string,
  room: string
): Promise<void> {
  const timestamp = Date.now();
  const historyKey = `queue:called:history:${queueType}`;
  await this.redis
    .multi()
    .hset(`queue:called:${queueType}:${patientId}`, {
      name,
      room,
      timestamp: timestamp.toString(),
    })
    .zadd(historyKey, timestamp, patientId.toString())
    .zremrangebyrank(historyKey, 0, -7)
    .exec();
}

async getLastCalledPatients(
  queueType: QueueType
): Promise<IQueueHistoryResponse[]> {
  const redis = this.getRedisClient();
  const limit = 6;
  const historyKey = `queue:called:history:${queueType}`;
}

```

```

const patientIds = await redis.zrevrange(historyKey, 0, limit - 1);

const patients: IQueueHistoryResponse[] = [];

for (const patientId of patientIds) {
  const data = await redis.hgetall(
    `queue:called:${queueType}:${patientId}`
  );
  if (data && Object.keys(data).length > 0) {
    patients.push({
      name: data.name,
      room: data.room,
      timestamp: Number(data.timestamp),
    });
  }
}

return patients;
}

async getQueuePatients(
  queueType: QueueType,
  start: number,
  end: number
): Promise<string[]> {
  const redis = this.getRedisClient();
  const queueKey = `queue:${queueType}`;
  return await redis.zrange(queueKey, start, end);
}

async getQueueLength(queueType: QueueType): Promise<number> {
  const redis = this.getRedisClient();
  const queueKey = `queue:${queueType}`;
  return await redis.zcard(queueKey);
}

async getNextPatientId(queueType: QueueType): Promise<string | null> {
  const redis = this.getRedisClient();
  const queueKey = `queue:${queueType}`;

  const ids = await redis.zrange(queueKey, 0, 0);
  return ids.length > 0 ? ids[0] : null;
}

```

```

async getPatientPosition(queueType: QueueType, patientId: number) {
  const queueKey = `queue:${queueType}`;
  const rank = await this.redis.zrank(queueKey, patientId.toString());
  return rank !== null ? rank + 1 : 0;
}

async getLastTimestamps(queueType: QueueType, limit = 10) {
  const historyKey = `queue:history:${queueType}`;
  const timestamps = await this.redis.zrange(historyKey, -limit, -1);
  return timestamps.map((t) => Number(t));
}

async setPatientStatus(patientId: number, status: QueueStatus) {
  const redis = this.getRedisClient();
  await redis.hset(`patient:${patientId}:queueData`, { status });
}

async registerPatientCalled(queueType: string, patientId: number) {
  const historyKey = `queue:history:${queueType}`;
  const timestamp = Date.now();
  await this.redis.zadd(
    historyKey,
    timestamp.toString(),
    patientId.toString()
  );
}

async patientExistsInQueue(patientId: number): Promise<boolean> {
  const meta = await this.getPatientMeta(patientId);
  return meta && Object.keys(meta).length > 0 && !meta.type;
}

```

TokenRepository.ts

```

import { RefreshToken } from "../database/models/RefreshToken";

export class TokenRepository {
  async saveRefreshToken(userId: number, token: string) {
    return RefreshToken.create({ userId, token });
  }

  async findRefreshToken(userId: number, token: string) {
    return RefreshToken.findOne({ where: { userId, token } });
  }
}

```

```

}

async findRefreshTokenByUser(userId: number) {
  return RefreshToken.findOne({ where: { userId } });
}

async saveOrUpdateRefreshToken(userId: number, newToken: string) {
  const existing = await this.findRefreshTokenByUser(userId);
  if (existing) {
    existing.token = newToken;
    await existing.save();
    return;
  }

  await this.saveRefreshToken(userId, newToken);
}

async deleteRefreshToken(userId: number, token: string) {
  return RefreshToken.destroy({ where: { usuario_id: userId, token } });
}

```

UserRepository.ts

```

import { Op } from "sequelize";
import User from "../database/models/User";
import { UserRole } from "../enums/User/UserRole";
import { ICreateUserInput } from "../interfaces/user/ICreateUser";
import { BadRequestError } from "../utils/errors/BadRequestError";

export class UserRepository {
  async create(data: ICreateUserInput, options?: any) {
    return User.create(
      {
        name: data.name,
        email: data.email,
        password: data.password,
        role: data.role,
      },
      options
    );
  }

  async findByEmail(email: string) {
    return User.findOne({ where: { email, deletado: false } });
  }
}

```

```

async findById(id: number) {
  return User.findOne({ where: { id, deletado: false } });
}

async findPaginated(params: {
  offset: number;
  limit: number;
  name?: string;
  role?: UserRole;
}) {
  const { offset, limit, name, role } = params;
  const whereClause: any = {};
  if (name) {
    whereClause.name = {
      [Op.like]: `%"${name}"`,
    };
  }
  if (role !== undefined) {
    whereClause.role = {
      [Op.eq]: role,
    };
  }
  const { count, rows } = await User.findAndCountAll({
    where: whereClause,
    limit,
    offset,
    order: [["data_criacao", "DESC"]],
  });
  return {
    users: rows,
    total: count,
  };
}

async delete(id: number) {
  const user = await this.findById(id);
  if (!user) throw new BadRequestError("Usuário não encontrado.");
  user.deleted = true;
  await user.save();
  return user;
}

async updateRole(newRole: UserRole, userId: number) {
  const user = await this.findById(userId);
  if (!user) throw new BadRequestError("Usuário não encontrado.");
}

```

```

    user.role = newRole;
    await user.save();
    return user;
}

async updateName(newName: string, userId: number) {
    const user = await this.findById(userId);
    if (!user) throw new BadRequestError("Usuário não encontrado.");
    user.name = newName;
    await user.save();
    return user;
}

async updatePassword(id: number, hashedPassword: string) {
    const user = await this.findById(id);
    if (!user) throw new BadRequestError("Usuário não encontrado.");
    user.password = hashedPassword;
    await user.save();
    return user;
}

async updateEmail(id: number, email: string) {
    const user = await this.findById(id);
    if (!user) throw new BadRequestError("Usuário não encontrado.");
    user.email = email;
    await user.save();
    return user;
}
}

```

WEB

App.tsx

```

import {
  BrowserRouter as Router,
  Routes,
  Route,
  Navigate,
} from "react-router-dom";
import { AuthProvider, useAuth } from "./context/authContext";
import { NotificationProvider } from "./components/notification/context";
import { ProtectedRoute } from "./components/protectedRoute";
import { GuestRoute } from "./components/guestRoute";

import LoginForm from "./pages/landingPage/pages/auth";
import ResetPassword from "./pages/landingPage/pages/forgotPass";

```

```

import PasswordSetupScreen from "./pages/landingPage/pages/resetPass";
import ConfirmEmailScreen from "./pages/landingPage/pages/confirmEmail";

import { Employee } from "./pages/employee/pages/home";
import { AddPatient } from "./pages/employee/pages/addPatient";

import { Dashboard } from "./pages/manager/pages/dashboard";
import { AdminsList } from "./pages/manager/pages/admins";
import { EmployeesList } from "./pages/manager/pages/employees";
import { Report } from "./pages/manager/pages/report";

import { ProfileScreen } from "./pages/general/pages/profile";
import type { JSX } from "react";
import EmployeeRegistrationScreen from "./pages/landingPage/pages/createEmployee";
import LogsList from "./pages/manager/pages/LogPage";
import { InRoom } from "./pages/employee/pages/inRoom";

type PublicRoute = {
  path: string;
  element: JSX.Element;
};

type ProtectedRoute = {
  path: string;
  element: JSX.Element;
  requiredRole?: number;
};

const ROLES = {
  ADMIN: 0,
  EMPLOYEE: 2,
} as const;

const routes: {
  public: PublicRoute[];
  protected: ProtectedRoute[];
} = {
  public: [
    { path: "/login", element: <LoginForm /> },
    { path: "/senha/esqueci", element: <ResetPassword /> },
    { path: "/email/confirmar", element: <ConfirmEmailScreen /> },
    { path: "/senha/redefinir", element: <PasswordSetupScreen /> },
    { path: "/registrar", element: <EmployeeRegistrationScreen /> },
  ],
};

```

```

protected: [
  { path: "/perfil", element: <ProfileScreen /> },
  {
    path: "/funcionario/filas/:id",
    element: <AddPatient />,
  },
  {
    path: "/funcionario/salas",
    element: <InRoom />,
    requiredRole: ROLES.EMPLOYEE,
  },
  {
    path: "/admin/funcionarios",
    element: <EmployeesList />,
    requiredRole: ROLES.ADMIN,
  },
  {
    path: "/admin/relatorio",
    element: <Report />,
    requiredRole: ROLES.ADMIN,
  },
  {
    path: "/admin/admins",
    element: <AdminsList />,
    requiredRole: ROLES.ADMIN,
  },
  { path: "/admin/logs", element: <LogsList />, requiredRole: ROLES.ADMIN },
  { path: "/admin", element: <Dashboard />, requiredRole: ROLES.ADMIN },
  {
    path: "/funcionario",
    element: <Employee />,
    requiredRole: ROLES.EMPLOYEE,
  },
],
};

function AppRoutes() {
  const { user, isAuthenticated } = useAuth();

  const getDefaultRoute = () => {
    if (!isAuthenticated || !user) {
      return "/login";
    }
  }
}

```

```

    return user.role === ROLES.ADMIN ? "/admin" : "/funcionario";
};

return (
<Routes>
{routes.public.map((route) => (
<Route
key={route.path}
path={route.path}
element={<GuestRoute>{route.element}</GuestRoute>}
/>
))}

{routes.protected.map((route) => (
<Route
key={route.path}
path={route.path}
element={
<ProtectedRoute requiredRole={route.requiredRole}>
{route.element}
</ProtectedRoute>
}
/>
))}

<Route path="/" element={<Navigate to={getOrDefaultRoute()} replace />} />

<Route path="*" element={<Navigate to="/" replace />} />
</Routes>
);
}

function App() {
return (
<AuthProvider>
<NotificationProvider>
<Router>
<AppRoutes />
</Router>
</NotificationProvider>
</AuthProvider>
);
}

```

```
export default App;
```

main.tsx

```
import { StrictMode } from "react";
import { createRoot } from "react-dom/client";
import "./index.css";
import App from "./App.tsx";

createRoot(document.getElementById("root")!).render(
  <StrictMode>
    <App />
  </StrictMode>
);
```

styles.ts

index.ts

```
import { color } from "@hisius/ui/theme/colors";
import { css, keyframes } from "styled-components";

const subtleSlide = keyframes`
from {
  opacity: 0;
  transform: translateY(10px);
}
to {
  opacity: 1;
  transform: translateY(0);
}
`;

const fadeIn = keyframes`
from {
  opacity: 0;
}
to {
  opacity: 1;
}
`;
```

```
const expandAnimation = keyframes`  
0% {  
  transform: scaleY(0.8);  
  opacity: 0.7;  
  max-height: 80px;  
}  
100% {  
  transform: scaleY(1);  
  opacity: 1;  
  max-height: 1000px;  
}  
`;  
  
const contractAnimation = keyframes`  
0% {  
  transform: scaleY(1);  
  opacity: 1;  
  max-height: 1000px;  
}  
100% {  
  transform: scaleY(0.98);  
  opacity: 0.9;  
  max-height: 80px;  
}  
`;  
  
export const tabAnim = css`  
  transform: translateX(5px);  
  color: ${color.primary};  
`;  
  
export const clickAnim = css`  
  &:active {  
    transform: scale(0.98);  
    transition: all 0.1s ease;  
  }  
`;  
  
export const expandAnimationAnim = css`  
  animation: ${expandAnimation} 0.4s ease forwards;  
`;  
  
export const contractAnimationAnim = css`
```

```

    animation: ${contractAnimation} 0.4s ease forwards;
';

export const fadeInAnim = css`
  animation: ${fadeIn} 0.4s ease;
`;

export const subtleSlideAnim = css`
  animation: ${subtleSlide} 0.4s ease;
`;

```

index.tsx

```

import { Container } from "./styles";
import { TbCopy } from "react-icons/tb";
interface CopyButtonProps {
  onClick?: () => void;
}

export function CopyButton({ onClick }: CopyButtonProps) {
  return (
    <Container onClick={onClick}>
      <TbCopy />
    </Container>
  );
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import styled from "styled-components";
import { clickAnim } from "../../assets/animations";

export const Container = styled.button`
  display: flex;
  justify-content: center;
  cursor: pointer;
  background: ${color.background};
  border: 0.7px solid rgba(13, 19, 41, 0.12);
  border-radius: 5px;

  & svg {
    width: 20px;
  }

```

```

    height: 20px;
}

${clickAnim}
';

```

index.tsx

```

import type { ReactNode } from "react";
import { Container } from "./styles";

interface FrontContainerProps {
  children?: ReactNode;
}

export function FrontContainer({ children }: FrontContainerProps) {
  return <Container>{children}</Container>;
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import styled from "styled-components";

export const Container = styled.div`
  position: relative;
  display: flex;
  width: 50vw;
  max-width: 535px;
  min-width: fit-content;
  padding: 1.5rem;
  background: ${color.front};
  border: 0.7px solid rgba(13, 19, 41, 0.12);
  border-radius: 5px;

  @media (max-width: 968px) {
    width: 80vw;
  }
`;

```

index.tsx

```

import { Navigate } from "react-router-dom";

```

```

import { useAuth } from "../../context/authContext";

interface GuestRouteProps {
  children: React.ReactNode;
}

export const GuestRoute: React.FC<GuestRouteProps> = ({ children }) => {
  const { user, isLoading, isAuthenticated } = useAuth();

  if (isLoading) {
    return (
      <div className="flex justify-center items-center min-h-screen">
        <div className="animate-spin rounded-full h-12 w-12 border-b-2 border-blue-500"></div>
      </div>
    );
  }

  if (isAuthenticated && user) {
    const redirectPath = user.role === 0 ? "/admin" : "/funcionario";
    return <Navigate to={redirectPath} replace />;
  }

  return <>{children}</>;
};

```

index.tsx

```

import { FaArrowLeft } from "react-icons/fa";
import { SearchPatient } from "../search";
import {
  ArrowContainer,
  Container,
  QueueContainer,
  QueueSubtitle,
  QueueTitle,
} from "./styles";
import { useNavigate } from "react-router-dom";
interface NavbarProps {
  onChange?: (value: string) => void;
  placeholder?: string;
  searchTerm?: string;
  onClick?: () => void;
  queueTitle: string;
  queueSubtitle: string;
  canSearch?: boolean;
}

```

```

    canBack?: boolean;
}

export function QueueHeader(props: NavbarProps) {
  const navigate = useNavigate();

  const handleBack = () => {
    navigate(-1);
  };

  return (
    <Container>
      <QueueContainer>
        {props.canBack ? (
          <ArrowContainer onClick={handleBack}>
            <FaArrowLeft />
          </ArrowContainer>
        ) : (
          ""
        )}
        <div>
          <QueueTitle>{props.queueTitle}</QueueTitle>
          <QueueSubtitle>{props.queueSubtitle}</QueueSubtitle>
        </div>
      </QueueContainer>
      {props.canSearch ? (
        <SearchPatient
          value={props.searchTerm}
          onChange={props.onChange}
          placeholder={props.placeholder}
          onClick={props.onClick}
        />
      ) : (
        ""
      )}
    </Container>
  );
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import styled from "styled-components";
import { subtleSlideAnim } from "../../assets/animations";

export const Container = styled.div`
  width: 100%;
  padding: 1rem 0rem;
  display: flex;

```

```
justify-content: space-between;
align-items: center;
margin-top: 1rem;
flex-wrap: wrap;
gap: 1rem;

${subtleSlideAnim}

@media (max-width: 877px) {
  flex-direction: column;
  align-items: stretch;
  gap: 1.5rem;
  padding: 0.5rem 0;
}

@media (max-width: 480px) {
  gap: 1rem;
  align-items: stretch;
  margin-top: 0.5rem;
}
';

export const ArrowContainer = styled.button`  

background: none;  

border: none;  

display: flex;  

justify-content: center;  

align-items: center;  

cursor: pointer;  

&:active {  

  transform: scale(0.98);  

  transition: all 0.1s ease;  

}  

& svg {  

  color: ${color.text};  

  width: 20px;  

  height: 40px;  

}  

`;
```

```

export const QueueContainer = styled.div`
  display: flex;
  align-items: center;
  gap: 2rem;
`;

export const QueueTitle = styled.h2`
  color: ${color.text};
  font-size: 24px;
  font-weight: 400;
  margin: 0;
`;

export const QueueSubtitle = styled.h3`
  color: ${color.text};
  font-size: 13px;
  font-weight: 200;
  margin: 0;
`;

```

container.tsx

```

import React from "react";
import { useNotification } from "./context";
import { NotificationContainerWrapper } from "./styles";
import { NotificationComponent } from ".";

export const NotificationContainer: React.FC = () => {
  const { notifications, removeNotification } = useNotification();

  if (notifications.length === 0) {
    return null;
  }

  return (
    <NotificationContainerWrapper>
      {notifications.map((notification) => (
        <NotificationComponent
          key={notification.id}
          notification={notification}
          onClose={() => removeNotification(notification.id)}
        />
      ))}
    </NotificationContainerWrapper>
  );
}

```

```
 );
};
```

context.tsx

```
import React, { createContext, useContext, useState, useCallback } from "react";
import type {
  NotificationContextType,
  NotificationProviderProps,
  Notification,
} from "./types";
import { NotificationContainer } from "./container";

const NotificationContext = createContext<NotificationContextType | undefined>(
  undefined
);

export const useNotification = (): NotificationContextType => {
  const context = useContext(NotificationContext);
  if (!context) {
    throw new Error(
      "useNotification deve ser usado dentro de um NotificationProvider"
    );
  }
  return context;
};

export const NotificationProvider: React.FC<NotificationProviderProps> = ({
  children,
}) => {
  const [notifications, setNotifications] = useState<Notification[]>([]);

  const addNotification = useCallback(
    (
      message: string,
      type: Notification["type"] = "info",
      duration: number = 5000
    ): string => {
      const id = Date.now() + Math.random().toString();
      const newNotification: Notification = {
        id,
        message,
        type,
        duration,
      };
      setNotifications((prevNotifications) => [...prevNotifications, newNotification]);
      return id;
    },
    []
  );

  return (
    <NotificationContext.Provider value={{ notifications, addNotification }}>
      {children}
    </NotificationContext.Provider>
  );
};
```

```

};

setNotifications((prev) => [...prev, newNotification]);

if (duration > 0) {
  setTimeout(() => {
    removeNotification(id);
  }, duration);
}

return id;
},
[]
);

const removeNotification = useCallback((id: string) => {
  setNotifications((prev) =>
    prev.filter((notification) => notification.id !== id)
  );
}, []);

const clearAll = useCallback(() => {
  setNotifications([]);
}, []);

const value: NotificationContextType = {
  notifications,
  addNotification,
  removeNotification,
  clearAll,
};

return (
  <NotificationContext.Provider value={value}>
    {children}
    <NotificationContainer />
  </NotificationContext.Provider>
);
};

```

index.tsx

```

import React, { useEffect, useState } from "react";
import type { NotificationComponentProps } from "./types";

```

```

import { TbSquareRoundedX } from "react-icons/tb";
import {
  NotificationWrapper,
  NotificationContent,
  NotificationMessage,
  CloseButton,
} from "./styles";

export const NotificationComponent: React.FC<NotificationComponentProps> = ({ notification, onClose, style }) => {
  const { message, type, duration } = notification;
  const [isExiting, setIsExiting] = useState<boolean>(false);

  const handleClose = () => {
    setIsExiting(true);
    setTimeout(() => {
      onClose();
    }, 250);
  };

  useEffect(() => {
    if (duration > 0) {
      const timer = setTimeout(() => {
        handleClose();
      }, duration - 300);

      return () => clearTimeout(timer);
    }
  }, [duration]);
}

return (
  <NotificationWrapper $type={type} $isExiting={isExiting} style={style}>
    <NotificationContent>
      <NotificationMessage>{message}</NotificationMessage>
      <CloseButton onClick={handleClose} aria-label="Fechar notificação">
        <TbSquareRoundedX />
      </CloseButton>
    </NotificationContent>
  </NotificationWrapper>
);
}

```

styles.ts

```

import styled, { keyframes, css } from "styled-components";
import type { NotificationWrapperProps } from "./types";
import { color } from "@hisius/ui/theme/colors";

const slideIn = keyframes`
from {
  transform: translateX(100%);
  opacity: 0;
}
to {
  transform: translateX(0);
  opacity: 1;
}
`;

const slideOut = keyframes`
from {
  transform: translateX(0);
  opacity: 1;
}
to {
  transform: translateX(100%);
  opacity: 0;
}
`;

export const NotificationContainerWrapper = styled.div`
position: fixed;
bottom: 20px;
right: 20px;
z-index: 1000;
display: flex;
flex-direction: column-reverse;
gap: 1rem;

@media (max-width: 768px) {
  right: 10px;
  left: 10px;
}
`;

```

```
export const NotificationWrapper = styled.div<NotificationWrapperProps>`  
  min-width: 300px;  
  max-width: 400px;  
  padding: 1rem;  
  
  font-weight: 300;  
  
  position: relative;  
  transition:  
    transform 0.3s ease,  
    opacity 0.3s ease;  
  
  border-left: 6px solid;  
  background-color: ${color.front};  
  
  border-radius: 5px;  
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);  
  
  animation: ${slideIn} 0.3s ease-out;  
  
  ${(props) => {  
    switch (props.$type) {  
      case "info":  
        return css`  
          border-color: ${color.primary};  
        `;  
      case "error":  
        return css`  
          border-color: ${color.error.error};  
        `;  
      case "warning":  
        return css`  
          border-color: ${color.error.warning};  
        `;  
      case "success":  
        return css`  
          border-color: ${color.error.ok};  
        `;  
      default:  
        return css`  
          border-color: ${color.primary};  
        `;  
    }  
  }}`
```

```
&:hover {
    transform: translateX(-5px);
}

${(props) =>
  props.$isExiting &&
  css`  

    animation: ${slideOut} 0.3s ease-in forwards;
`}

@media (max-width: 768px) {
  min-width: auto;
  max-width: none;
}
';

export const NotificationContent = styled.div`  

  display: flex;  

  justify-content: space-between;  

  align-items: flex-start;  

  gap: 1rem;
`;

export const NotificationMessage = styled.span`  

  font-size: 14px;  

  flex: 1;  

  margin: 0;
`;

export const CloseButton = styled.button`  

  background: none;  

  border: none;  

  cursor: pointer;  

  padding: 0;  

  width: 24px;  

  height: 24px;  

  display: flex;  

  align-items: center;  

  justify-content: center;  

  border-radius: 50%;  

  color: inherit;  

  opacity: 0.7;  

  transition:

```

```

  opacity 0.2s ease,
  background-color 0.2s ease;

&:hover {
  opacity: 1;
  background-color: rgba(0, 0, 0, 0.1);
}
';

```

types.ts

```

import type { ReactNode } from "react";

export type NotificationType = "info" | "error" | "warning" | "success";

export interface Notification {
  id: string;
  message: string;
  type: NotificationType;
  duration: number;
}

export interface NotificationContextType {
  notifications: Notification[];
  addNotification: (
    message: string,
    type?: NotificationType,
    duration?: number
  ) => string;
  removeNotification: (id: string) => void;
  clearAll: () => void;
}

export interface NotificationProviderProps {
  children: ReactNode;
}

export interface NotificationComponentProps {
  notification: Notification;
  onClose: () => void;
  style?: React.CSSProperties;
}

export interface NotificationWrapperProps {

```

```
$type: NotificationType;
$isExiting?: boolean;
}
```

index.tsx

```
import { Container, PageButton } from "./styles";

interface PaginationProps {
  totalItems: number;
  itemsPerPage?: number;
  currentPage?: number;
  onPageChange: (page: number) => void;
  maxVisibleButtons?: number;
}

export default function Pagination({
  totalItems,
  itemsPerPage = 10,
  currentPage = 1,
  onPageChange,
  maxVisibleButtons = 5,
}: PaginationProps) {
  const totalPages = Math.ceil(totalItems / itemsPerPage);

  const changePage = (newPage: number) => {
    if (newPage < 1 || newPage > totalPages || newPage === currentPage) {
      return;
    }
    onPageChange(newPage);
  };

  const generatePageNumbers = (): (number | string)[] => {
    if (totalPages <= 1) return [];

    const numbers: (number | string)[] = [];
    const half = Math.floor(maxVisibleButtons / 2);

    let start = Math.max(1, currentPage - half);
    let end = Math.min(totalPages, start + maxVisibleButtons - 1);

    if (end - start + 1 < maxVisibleButtons) {
      start = Math.max(1, end - maxVisibleButtons + 1);
    }
  };
}
```

```

if (start > 1) {
    numbers.push(1);
    if (start > 2) numbers.push("...");
}

for (let i = start; i <= end; i++) {
    numbers.push(i);
}

if (end < totalPages) {
    if (end < totalPages - 1) numbers.push("...");
    numbers.push(totalPages);
}

return numbers;
};

if (totalPages <= 1) return null;

const pageNumbers = generatePageNumbers();

return (
    <Container>
        {/* Botão anterior */}
        <PageButton
            onClick={() => changePage(currentPage - 1)}
            disabled={currentPage === 1}
        >
            <
        </PageButton>

        {/* Números das páginas */}
        {pageNumbers.map((number, index) => (
            <PageButton
                key={index}
                $active={number === currentPage}
                $dots={number === "..."}
                onClick={() => typeof number === "number" && changePage(number)}
                disabled={number === "..."}
            >
                {number}
            </PageButton>
        ))}
    )
);

```

```

/* Botão próximo */
<PageButton
  onClick={() => changePage(currentPage + 1)}
  disabled={currentPage === totalPages}
>
  >
</PageButton>
</Container>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import styled from "styled-components";
import { clickAnim } from "../../assets/animations";

export const Container = styled.div`
  display: flex;
  justify-content: center;
  gap: 0.3rem;
  margin: 1rem 0;
  flex-wrap: wrap;
`;

export const PageButton = styled.button` ${$active?: boolean; $dots?: boolean }>
  padding: 0.5rem 0.8rem;
  border: 1px solid ${color.gray};
  background-color: white;
  color: ${color.text};
  cursor: pointer;
  border-radius: 4px;
  transition: all 0.2s ease;
  min-width: 2.5rem;
  font-family: inherit;

  ${clickAnim}

  &:hover {
    background-color: ${color.front};
    border-color: ${color.primary};
  }
` ${({props}) =>
  props.$active &&
  `background-color: ${color.primary}`}

```

```

color: white;
border-color: ${color.primary}

&:hover {
  background-color: ${color.primary}
  border-color: ${color.primary}
}

${(props) =>
  props.$dots &&
  background-color: transparent;
  border: none;
  cursor: default;

  &:hover {
    background-color: transparent;
    border: none;
  }
}

&:disabled {
  cursor: not-allowed;
  opacity: 0.6;
}
;

```

index.tsx

```

import React, { useEffect } from "react";
import {
  PopupOverlay,
  PopupContainer,
  PopupHeader,
  PopupTitle,
  CloseButton,
  PopupContent,
} from "./styles";
import type { PopupProps } from "./types";
import { TbSquareRoundedX } from "react-icons/tb";

const Popup: React.FC<PopupProps> = ({  

  isOpen = false,  

  onClose,  

  title,  

  children,

```

```

showCloseButton = true,
closeOnOverlayClick = true,
closeOnEscape = true,
size = "medium",
position = "center",
className,
}) => {
useEffect(() => {
if (!closeOnEscape || !isOpen) return;

const handleEscape = (e: KeyboardEvent) => {
if (e.key === "Escape") {
onClose();
}
};

document.addEventListener("keydown", handleEscape);
return () => document.removeEventListener("keydown", handleEscape);
}, [isOpen, closeOnEscape, onClose]);

useEffect(() => {
document.body.style.overflow = isOpen ? "hidden" : "unset";

return () => {
document.body.style.overflow = "unset";
};
}, [isOpen]);

const handleOverlayClick = (e: React.MouseEvent<HTMLDivElement>) => {
if (e.target === e.currentTarget && closeOnOverlayClick) {
onClose();
}
};

if (!isOpen) return null;

return (
<PopupOverlay
onClick={handleOverlayClick}
className={className}
position={position}
>
<PopupContainer size={size}>
{!(title || showCloseButton) && (

```

```

<PopupHeader>
  {showCloseButton && (
    <CloseButton onClick={onClose} aria-label="Fechar">
      <TbSquareRoundedX />
    </CloseButton>
  )}
</PopupHeader>
)}
{title && <PopupTitle>{title}</PopupTitle>}
<PopupContent>{children}</PopupContent>
</PopupContainer>
</PopupOverlay>
);
};
};

export default Popup;

```

styles.ts

```

import styled from "styled-components";
import type { PopupOverlayProps, PopupContainerProps } from "./types";
import {
  clickAnim,
  fadeInAnim,
  subtleSlideAnim,
} from "../../assets/animations";

export const PopupOverlay = styled.div<PopupOverlayProps>`
  position: fixed;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: rgba(0, 0, 0, 0.6);
  display: flex;
  justify-content: center;
  align-items: ${({ position }) =>
    position === "top"
      ? "flex-start"
      : position === "bottom"
        ? "flex-end"
        : "center"};
  padding: ${({ position }) =>
    position === "top" ? "20px" : position === "bottom" ? "20px" : "0"};

```

```
z-index: 1000;
${fadeInAnim}
';

export const PopupContainer = styled.div<PopupContainerProps>'  
background: white;  
border-radius: 12px;  
box-shadow: 0 10px 30px rgba(0, 0, 0, 0.3);  
max-height: 90vh;  
overflow: hidden;  
display: flex;  
flex-direction: column;  
${subtleSlideAnim}

${({ size }) => {
  switch (size) {
    case "small":
      return `  
width: 400px;  
max-width: 90vw;  
`;
    case "large":
      return `  
width: 800px;  
max-width: 90vw;  
`;
    case "fullscreen":
      return `  
width: 95vw;  
height: 95vh;  
`;
    default:
      return `  
width: 600px;  
max-width: 90vw;  
`;
  }
}}
```

';

```
export const PopupHeader = styled.div'  
display: flex;  
justify-content: end;  
align-items: center;
```

```

padding: 1rem;
';

export const PopupTitle = styled.h2`
width: 100%;
text-align: center;
margin: 0;
padding: 0 3rem;
font-size: 1.5rem;
font-weight: 400;
margin-bottom: 1rem;
`;

export const CloseButton = styled.button`
background: none;
border: none;

cursor: pointer;
padding: 0;

width: 40px;
height: 40px;

display: flex;
align-items: center;
justify-content: center;

${clickAnim}
`;

```

export const PopupContent = styled.div`
padding: 24px;
overflow-y: auto;
flex: 1;
`;

types.ts

```

export interface PopupProps {
isOpen?: boolean;
onClose: () => void;
title?: string;
children: React.ReactNode;
showCloseButton?: boolean;

```

```

closeOnOverlayClick?: boolean;
closeOnEscape?: boolean;
size?: "small" | "medium" | "large" | "fullscreen";
position?: "center" | "top" | "bottom";
className?: string;
}

```

```

export interface PopupOverlayProps {
  position: "center" | "top" | "bottom";
}

```

```

export interface PopupContainerProps {
  size: "small" | "medium" | "large" | "fullscreen";
}

```

index.tsx

```

import { Navigate } from "react-router-dom";
import { useAuth } from "../../context/authContext";

interface ProtectedRouteProps {
  children: React.ReactNode;
  requiredRole?: number;
}

export const ProtectedRoute: React.FC<ProtectedRouteProps> = ({ 
  children,
  requiredRole,
}) => {
  const { user, isAuthenticated, isLoading } = useAuth();

  if (isLoading) {
    return (
      <div className="flex justify-center items-center h-screen">
        <div>Carregando...</div>
      </div>
    );
  }

  if (!isAuthenticated || !user) {
    return <Navigate to="/login" replace />;
  }

  if (requiredRole !== undefined && user.role !== requiredRole) {

```

```

const defaultRoute = user.role === 0 ? "/admin" : "/funcionario";
return <Navigate to={defaultRoute} replace />;
}

return <>{children}</>;
};

```

index.tsx

```

import type { ChangeEvent } from "react";
import { SearchInput, SearchContainer, IconContainer } from "./styles";
import { FaSearch } from "react-icons/fa";

interface SearchPatientProps {
  value?: string;
  onChange?: (value: string) => void;
  placeholder?: string;
  onClick?: () => void;
}

export function SearchPatient({
  value,
  onChange,
  placeholder,
  onClick,
}: SearchPatientProps) {
  return (
    <SearchContainer>
      <SearchInput
        type="text"
        id="search"
        value={value}
        onChange={(e: ChangeEvent<HTMLInputElement>) =>
          onChange?.(e.target.value)
        }
        placeholder={placeholder}
      />
      {onClick ? (
        <IconContainer onClick={onClick}>
          <FaSearch />
        </IconContainer>
      ) : (
        ""
      )}
    
```

```

    </SearchContainer>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import styled from "styled-components";
import { clickAnim } from "../../assets/animations";

export const SearchContainer = styled.div`
  position: relative;
  display: flex;
  min-width: 500px;
  background: ${color.front};
  border: 0.7px solid rgba(13, 19, 41, 0.12);
  border-radius: 5px;

  @media (max-width: 877px) {
    min-width: 200px;
  }
`;

export const IconContainer = styled.button`
  width: 10%;
  display: flex;
  align-items: center;
  justify-content: center;
  background: none;
  border: none;

  cursor: pointer;

  ${clickAnim}

  & svg {
    color: ${color.text};
  }
`;

export const SearchInput = styled.input`
  flex: 1;
  padding: 15px 20px;
  border: none;

```

```

border-radius: 5px;
font-size: 13px;
background: none;
transition: all 0.2s;

&:focus {
  outline: none;
  border-color: ${color.primary};
  box-shadow: 0 0 0 3px rgba(59, 130, 246, 0.1);
}

&::placeholder {
  color: ${color.text};
  font-weight: 200;
}
';

```

index.tsx

```

import { capitalizeWords } from "../../utils";
import {
  SelectContainer,
  SelectLabel,
  SelectComponent,
  ErrorMessage,
  Options,
} from "./styles";

interface SelectProps {
  label?: string;
  error?: string;
  value?: string;
  options: { value: string; label: string }[];
  onClick?: (event: React.MouseEvent) => void;
  onChange?: (event: React.ChangeEvent<HTMLSelectElement>) => void;
  placeholder?: string;
  disabled?: boolean;
  required?: boolean;
}

export function Select({
  label,
  error,

```

```
options,
onClick,
onChange,
value,
placeholder = "Selecione uma opção",
disabled = false,
required = false,
}: SelectProps) {
const handleSelectClick = (event: React.MouseEvent) => {
  if (disabled) return;
  event.stopPropagation();
  onClick?.(event);
};

const handleChange = (event: React.ChangeEvent<HTMLSelectElement>) => {
  if (disabled) return;
  onChange?.(event);
};

return (
  <SelectContainer>
    {label && (
      <SelectLabel>
        {label}
        {required && (
          <span style={{ color: "#e53e3e", marginLeft: "4px" }}>*</span>
        )}
      </SelectLabel>
    )}
    <SelectComponent
      value={value}
      onChange={handleChange}
      onClick={handleSelectClick}
      $hasError={!error}
      disabled={disabled}
      required={required}
      aria-invalid={!error}
      aria-describedby={error ? `error-${label}` : undefined}
    >
      <Options value="" disabled hidden>
        {placeholder}
      </Options>
    
```

{options.map((option) => (

```

<Options key={option.value} value={option.value}>
  {capitalizeWords(option.label)}
</Options>
))}
</SelectComponent>
{error && <ErrorMessage id={`error-${label}`}>{error}</ErrorMessage>}
</SelectContainer>
);
}

```

styles.ts

```
import { color } from "@hisius/ui/theme/colors";
import styled from "styled-components";
```

```
interface SelectProps {
  $hasError?: boolean;
}
```

```
export const SelectContainer = styled.div`
  display: flex;
  flex-direction: column;
  gap: 6px;
  width: 100%;
  max-width: 300px;
```

```
@media (max-width: 768px) {
  max-width: 100%;
  gap: 4px;
}
`;
```

```
export const SelectLabel = styled.label`
  font-size: 13px;
  color: ${color.text};
  margin-bottom: 4px;
  font-weight: 500;
```

```
@media (max-width: 768px) {
  font-size: 14px;
  margin-bottom: 2px;
}
```

```
@media (max-width: 480px) {
```

```

    font-size: 13px;
}
';

export const Options = styled.option`  

  font-size: 13px;  

  font-weight: 400;  

  padding: 8px;  

  @media (max-width: 480px) {  

    font-size: 14px;  

    padding: 10px;  

  }
';
`;

export const SelectComponent = styled.select<SelectProps>`  

  background-color: ${color.background};  

  border: 1.5px solid  

    ${(props) => (props.$hasError ? color.error.error : color.gray)};  

  border-radius: 4px;  

  padding: 0.75rem;  

  font-size: 13px;  

  font-weight: 400;  

  color: ${color.text};  

  width: 100%;  

  cursor: pointer;  

  transition: all 0.25s ease;  

  appearance: none;  

  background-image: url("data:image/svg+xml,%3Csvg xmlns='http://www.w3.org/2000/svg'  

width='16' height='16' viewBox='0 0 24 24' fill='none' stroke='%236B7280' stroke-width='2'  

stroke-linecap='round' stroke-linejoin='round'%3E%3Cpolyline points='6 9 12 15 18  

9%3E%3C/polyline%3E%3C/svg%3E");  

  background-repeat: no-repeat;  

  background-position: right 12px center;  

  background-size: 16px;  

  @media (max-width: 768px) {  

    font-size: 14px;  

    padding: 0.875rem;  

    background-position: right 16px center;  

  }
  

  @media (max-width: 480px) {

```

```
font-size: 16px;
padding: 1rem;
background-size: 18px;
background-position: right 14px center;
}

&:focus {
  outline: none;
  border-color: ${(props) =>
    props.$hasError ? color.error.error : color.secondary};
  box-shadow: 0 0 0 3px
  ${(props) =>
    props.$hasError ? "rgba(229, 62, 62, 0.1)" : "rgba(0, 123, 255, 0.1)"};
}

&:active {
  transform: scale(0.98);
  transition: all 0.1s ease;
}

&:hover {
  border-color: ${(props) =>
    props.$hasError ? color.error.error : color.secondary};
}

&::placeholder {
  color: ${color.gray};
}

@media (max-width: 480px) {
  &:active {
    transform: none;
  }
}

&:disabled {
  background-color: ${color.deactive};
  cursor: not-allowed;
  opacity: 0.6;
}

export const ErrorMessage = styled.span`  

  font-size: 13px;
```

```

color: ${color.error.error};
margin-top: 4px;
font-weight: 400;
display: flex;
align-items: center;
gap: 4px;

@media (max-width: 768px) {
  font-size: 12px;
  margin-top: 2px;
}

@media (max-width: 480px) {
  font-size: 11px;
}

&::before {
  content: "⚠";
  font-size: 12px;
}
';

export const SelectWrapper = styled.div` 
  display: flex;
  flex-direction: column;
  width: 100%;

  @media (min-width: 769px) {
    align-items: flex-start;
  }

  @media (max-width: 768px) {
    align-items: stretch;
  }
`;

```

index.tsx

```

import { useState, type ReactNode } from "react";
import { useLocation, useNavigate } from "react-router-dom";
import {
  BottomSection,
  ExpandButton,

```

```
LogoImage,
LogoSection,
LogoTitle,
MenuItemIcon,
MenuItem,
MenuList,
MenuSection,
MenuItemText,
SidebarBackground,
SidebarContainer,
} from "./styles";
import logo from "@hisius/ui/assets/images/logo.png";
import { TbLayoutSidebarLeftExpand } from "react-icons/tb";

interface Routes {
  icon: ReactNode;
  text: string;
  path: string;
  exact?: boolean;
}

interface SidebarProps {
  menuItems: Routes[];
}

export default function Sidebar({ menuItems }: SidebarProps) {
  const [isExpanded, setIsExpanded] = useState(false);
  const navigate = useNavigate();

  const handleExpand = () => {
    setIsExpanded(!isExpanded);
  };

  const handleMenuItemClick = (path: string) => {
    navigate(path);
  };

  const location = useLocation();

  const isActive = (path: string, exact?: boolean) => {
    const currentPath = location.pathname;

    if (exact) {
      return path === currentPath;
    }
  };
}
```

```

    }

    return currentPath.startsWith(path + "/") || path === currentPath;
};

return (
  <>
  <SidebarBackground $isExpanded={isExpanded} />
  <SidebarContainer $isExpanded={isExpanded}>
    <LogoSection
      $isExpanded={isExpanded}
      onClick={() => navigate("/")}
      style={{ cursor: "pointer" }}
    >
      <LogoImage src={logo} alt="Logo" />
      <LogoTitle $isExpanded={isExpanded}>Hisius</LogoTitle>
    </LogoSection>

    <MenuSection>
      <MenuList>
        {menuItems.map((item, index) => (
          <MenuItem
            key={index}
            $isExpanded={isExpanded}
            className={isActive(item.path, item.exact) ? "active" : ""}
            onClick={() => handleMenuItemClick(item.path)}
          >
            <MenuIcon $isExpanded={isExpanded}>{item.icon}</MenuIcon>
            <MenuText $isExpanded={isExpanded}>{item.text}</MenuText>
          </MenuItem>
        )));
      </MenuList>
    </MenuSection>

    <BottomSection>
      <ExpandButton onClick={handleExpand} $isExpanded={isExpanded}>
        <TbLayoutSidebarLeftExpand />
      </ExpandButton>
    </BottomSection>
  </SidebarContainer>
</>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import styled from "styled-components";

export const SidebarBackground = styled.div`$isExpanded: boolean >` {
  position: absolute;
  left: 0;
  top: 0;

  width: 100vw;
  height: 100vh;

  z-index: 10;

  transition: all ease 0.4s;

  display: ${props => (props.$isExpanded ? "block" : "none")};

  background-color: ${props =>
    props.$isExpanded ? "rgba(0, 0, 0, 0.5)" : "rgba(0, 0, 0, 0)"};
}

export const LogoTitle = styled.h1`$isExpanded: boolean >` {
  font-size: 16px;
  font-weight: 400;
  text-transform: uppercase;
  display: ${props => (props.$isExpanded ? "block" : "none")};
}

export const SidebarContainer = styled.div`$isExpanded: boolean >` {
  width: ${props => (props.$isExpanded ? "180px" : "70px")};
  height: 100vh;
  background-color: ${color.front};
  color: ${color.text};
  position: fixed;
  left: 0;
  top: 0;
  display: flex;
  flex-direction: column;
  transition: width 0.3s ease;
  overflow: hidden;
  z-index: 11;
}

export const LogoImage = styled.img`$isExpanded: boolean >` {
  width: 30px;
  aspect-ratio: 1/1;
}

```

```
export const LogoSection = styled.div`$isExpanded: boolean >`  
padding: 20px;  
display: flex;  
gap: 1rem;  
align-items: center;  
font-size: ${props => (props.$isExpanded ? "24px" : "16px")};  
font-weight: bold;  
white-space: nowrap;  
text-align: ${props => (props.$isExpanded ? "left" : "center")};  
`;  
  
export const MenuSection = styled.div`  
flex: 1;  
padding: 20px 0;  
`;  
  
export const BottomSection = styled.div`  
padding: 10px;  
display: flex;  
justify-content: flex-end;  
`;  
  
export const ExpandButton = styled.button`$isExpanded: boolean >`  
width: fit-content;  
padding: 10px;  
color: ${color.text};  
border: none;  
border-radius: 4px;  
cursor: pointer;  
transition: background-color 0.2s;  
white-space: nowrap;  
  
&:hover {  
background-color: ${color.gray};  
}  
`;  
  
export const MenuList = styled.ul`  
list-style: none;  
padding: 0;  
margin: 0;  
`;  
  
export const MenuItem = styled.li`$isExpanded: boolean >`  
padding: 12px;  
cursor: pointer;  
font-size: 13px;  
transition: background-color 0.2s;  
display: flex;  
align-items: center;
```

```

justify-content: ${(props) => (props.$isExpanded ? "flex-start" : "center")};

&:hover {
  background-color: ${color.gray}88;
}

&.active {
  background-color: ${color.gray};
  & * {
    font-weight: 500;
  }
}
';

export const MenuIcon = styled.span<{ $isExpanded: boolean }>`  

& svg {
  height: 15px;
  width: 15px;
}  

margin-right: ${(props) => (props.$isExpanded ? "20px" : "0")};  

`;

export const MenuText = styled.span<{ $isExpanded: boolean }>`  

font-size: 13px;  

white-space: nowrap;  

opacity: ${(props) => (props.$isExpanded ? 1 : 0)};  

display: ${(props) => (props.$isExpanded ? "block" : "none")};  

transition: opacity 0.2s;  

`;

export const MainContent = styled.main<{ $isExpanded: boolean }>`  

margin-left: ${(props) => (props.$isExpanded ? "250px" : "60px")};  

padding: 40px;  

min-height: 100vh;  

background-color: #ecf0f1;  

transition: margin-left 0.3s ease;  

`;

```

index.tsx

```

import type { ReactNode } from "react";
import { Container, Title, Value } from "./styles";

interface TextValueProps {
  title: string;
  children: ReactNode;
}

export function TextValue(props: TextValueProps) {

```

```

return (
  <Container>
    <Title>{props.title}</Title>
    <Value>{props.children}</Value>
  </Container>
);
}

```

styles.ts

```
import styled from "styled-components";
```

```
export const Container = styled.div`
  width: fit-content;
  gap: 1rem;
  display: flex;
  justify-content: space-between;
`;
```

```
export const Title = styled.span`
  font-size: 13px;
  font-weight: 200;
`;
```

```
export const Value = styled.span`
  font-size: 13px;
  font-weight: 400;
`;
```

authContext.tsx

```
import LocalStorageManager from "@hisius/services/src/helpers/localStorageManager";
import type { User } from "@hisius/interfaces/src";
import React, {
  createContext,
  useContext,
  useState,
  useEffect,
  type ReactNode,
} from "react";
```

```
interface AuthContextType {
  user: User | null;
}
```

```

accessToken: string | null;
login: (userData: User, token: string) => void;
logout: () => void;
isLoading: boolean;
isAuthenticated: boolean;
}

const AuthContext = createContext<AuthContextType | undefined>(undefined);

export const AuthProvider: React.FC<{ children: ReactNode }> = ({

  children,
}) => {
  const [user, setUser] = useState<User | null>(null);
  const [accessToken, setAccessToken] = useState<string | null>(null);
  const [isLoading, setIsLoading] = useState(true);

  useEffect(() => {
    const savedUser = LocalStorageManager.getUser();
    const savedToken = LocalStorageManager.getAccessToken();

    if (savedUser && savedToken) {
      setUser(savedUser);
      setAccessToken(savedToken);
    }
  }

  setIsLoading(false);
}, []);

const login = (userData: User, token: string) => {
  setUser(userData);
  setAccessToken(token);
  LocalStorageManager.setUser(userData);
  LocalStorageManager.setTokens(token);
};

const logout = () => {
  setUser(null);
  setAccessToken(null);
  LocalStorageManager.clearAuth();
};

const isAuthenticated = !!user && !!accessToken;

return (

```

```

<AuthContext.Provider
  value={ {
    user,
    accessToken,
    login,
    logout,
    isLoading,
    isAuthenticated,
  } }
>
  {children}
</AuthContext.Provider>
);
};

export const useAuth = () => {
  const context = useContext(AuthContext);
  if (context === undefined) {
    throw new Error("useAuth must be used within an AuthProvider");
  }
  return context;
};

```

index.tsx

```

import type { ApiError, ApiResponse } from "@hisius/interfaces/src";
import { useState, useCallback } from "react";

export const useFormErrors = () => {
  const [errors, setErrors] = useState<Record<string, string>>({});

  const setFieldError = useCallback((field: string, message: string) => {
    setErrors((prev) => ({ ...prev, [field]: message }));
  }, []);

  const clearFieldError = useCallback((field: string) => {
    setErrors((prev) => {
      const newErrors = { ...prev };
      delete newErrors[field];
      return newErrors;
    });
  }, []);

  const clearAllErrors = useCallback(() => {

```

```

    setErrors({});  

}, []);  
  

const handleApiErrors = useCallback((errorResponse: ApiResponse) => {  

  const newErrors: Record<string, string> = {};  
  

  if (errorResponse.errors?.length) {  

    errorResponse.errors.forEach((error: ApiError) => {  

      newErrors[error.field] = error.message;  

    });  

  }  
  

  setErrors(newErrors);  

}, []);  
  

return {  

  errors,  

  setFieldError,  

  clearFieldError,  

  clearAllErrors,  

  handleApiErrors,  

};  

};  


```

index.tsx

```

import { useEffect } from "react";  
  

export function usePageTitle(title: string) {  

  useEffect(() => {  

    document.title = title;  

  }, [title]);  

}

```

index.ts

```

import { useState, useRef, useEffect } from "react";  
  

export function useTruncate() {  

  const [isTruncated, setIsTruncated] = useState(false);  

  const ref = useRef<HTMLDivElement>(null);  
  

  useEffect(() => {  


```

```

const checkTruncation = () => {
  if (ref.current) {
    const element = ref.current;
    setIsTruncated(element.scrollWidth > element.clientWidth);
  }
};

checkTruncation();
window.addEventListener("resize", checkTruncation);

return () => window.removeEventListener("resize", checkTruncation);
}, []);

return { ref, isTruncated };
}

```

index.tsx

```

import { HiOutlineQueueList } from "react-icons/hi2";
import { HiOutlineUser } from "react-icons/hi2";
import SidebarComponent from "../../../../../components/sidebar";
import { HiOutlineBuildingOffice2 } from "react-icons/hi2";

export function Sidebar() {
  const menuItems = [
    {
      icon: <HiOutlineQueueList />,
      text: "Filas",
      path: "/funcionario",
      exact: true,
    },
    {
      icon: <HiOutlineBuildingOffice2 />,
      text: "Salas",
      path: "/funcionario/salas",
      exact: true,
    },
    {
      icon: <HiOutlineUser />,
      text: "Conta",
      path: "/perfil",
      exact: true,
    },
  ];
}

```

```

return (
  <>
  <SidebarComponent menuItems={menuItems} />
  </>
);
}

```

styles.ts

index.tsx

```

import { useState } from "react";
import { Switch, Slider, LeftText, RightText } from "./styles";

interface ToggleProps {
  initial?: boolean;
  onToggle?: (state: boolean) => void;
  labels?: {
    on: string;
    off: string;
  };
}

export default function Toggle({
  initial = false,
  onToggle,
  labels = { on: "Sim", off: "Não" },
}: ToggleProps) {
  const [isOn, setIsOn] = useState(initial);

  const handleClick = () => {
    const newState = !isOn;
    setIsOn(newState);
    onToggle?.(newState);
  };

  return (
    <Switch onClick={handleClick} isOn={isOn}>
      <Slider isOn={isOn} />
      <LeftText isOn={isOn}>{labels.off}</LeftText>
      <RightText isOn={isOn}>{labels.on}</RightText>
    
```

```

        </Switch>
    );
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import { clickAnim } from "../../../../../assets/animations";
import styled, { css, keyframes } from "styled-components";

interface StyledProps {
    isOn: boolean;
    position?: "left" | "right";
}

const bounceIn = keyframes`
  0% { transform: scale(0.95); }
  50% { transform: scale(1.02); }
  100% { transform: scale(1); }
`;

export const Switch = styled.div<StyledProps>`
    display: inline-grid;
    grid-template-columns: 1fr 1fr;
    align-items: center;
    background: ${color.front};
    border-radius: 2.5px;
    position: relative;
    cursor: pointer;
    padding: 4px;
    border: 0.7px solid rgba(13, 19, 41, 0.12);
    overflow: hidden;
    min-width: 120px;
    ${clickAnim}
`;

```

```

export const Slider = styled.div<StyledProps>`
    width: calc(50% - 8px);
    height: calc(100% - 8px);
    background: ${color.primary};
    border-radius: 5px;
    position: absolute;
    top: 4px;
    left: ${({props: StyledProps}) => (props.isOn ? "calc(50% + 2px)" : "4px")};
    transition: all 0.3s ease;
`;

```

```

    box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);

    ${(props: StyledProps) =>
      props.isOn !== undefined &&
      css`  

        animation: ${bounceIn} 0.4s cubic-bezier(0.68, -0.55, 0.265, 1.55);
      `}
    `;

export const Text = styled.span<StyledProps>
  color: ${(props: StyledProps) =>
    props.position === "left"
      ? props.isOn
        ? color.text
        : color.front
      : props.isOn
        ? color.front
        : color.text};

  font-size: 14px;
  font-weight: 500;
  transition: all 0.4s cubic-bezier(0.34, 1.56, 0.64, 1);
  z-index: 1;
  text-align: center;
  padding: 8px 12px;
  white-space: nowrap;
  position: relative;

  transform: ${(props: StyledProps) =>
    (props.position === "left" && !props.isOn) ||
    (props.position === "right" && props.isOn)
      ? "scale(1.05)"
      : "scale(1)"};
  `;

export const LeftText = styled(Text).attrs({ position: "left" })``;
export const RightText = styled(Text).attrs({ position: "right" })``;

```

index.tsx

```

import { ButtonContainer, Container, ContentContainer } from "./styles";
import { QueueHeader } from "../../../../../components/navbar";
import { AddCard } from "./components/addCard";
import CustomButton from "@hisius/ui/components/Button";

```

```

import { Sidebar } from "../../components/sidebar";
import { useEffect, useState } from "react";
import { useNavigate, useParams } from "react-router-dom";
import type { IPatient } from "@hisius/interfaces/src";
import { Queue } from "@hisius/services/src";
import { useNotification } from "../../../../../components/notification/context";
import { usePageTitle } from "../../../../../hooks/PageTitle";

export function AddPatient() {
  const { id } = useParams<{ id: string }>();
  const [patientData, setPatientData] = useState<IPatient | null>(null);
  const [selectedClassification, setSelectedClassification] =
    useState<string>("");
  const { addNotification } = useNotification();
  const navigate = useNavigate();
  const queueService = new Queue();

  usePageTitle("Cadastrar na fila - Hisius");

  useEffect(() => {
    const fetchPatientData = async () => {
      if (!id) {
        addNotification("Id do paciente não encontrado", "error");
        return;
      }
      try {
        const patient = await queueService.getPatient(Number(id));
        setPatientData(patient);
      } catch (error) {
        addNotification("Paciente não encontrado", "error");
        navigate("/funcionario");
      }
    };
    fetchPatientData();
  }, [id]);

  const handleClassificationChange = (classification: string) => {
    setSelectedClassification(classification);
  };

  const handleCallNext = async () => {
    try {
      await queueService.goToTreatment(

```

```
Number(patientData?.id),
selectedClassification
);

addNotification("Paciente foi movido para a outra fila", "success");
navigate("/funcionario");
} catch (err: any) {
const errors = err.response?.data?.errors || [];
const messages = errors
.map((error: any) => error?.message || error)
.filter(Boolean);

if (messages.length === 0) {
messages.push(
err.response?.data?.message || err.message || "Erro ao mover paciente"
);
}

messages.forEach((message: string) => {
addNotification(message, "error");
});

return (
<>
<Sidebar />
<Container className="containerSide">
<QueueHeader
queueTitle="Cadastrar na fila"
queueSubtitle="Passar para a fila de atendimento"
canBack
/>
<ContentContainer>
{patientData && (
<AddCard
name={patientData.name || "Nome não informado"}
patient={
{
id: patientData.id,
age: patientData.age || 0,
gender: patientData.gender || "Sexo não informado",
birthDate: patientData.birthDate || "",
motherName: patientData.motherName || "Não informado",
})}

```

```

        cnsNumber: patientData.cnsNumber || "Não informado",
    } as IPatient
    }
    selectedClassification={selectedClassification}
    onClassificationChange={handleClassificationChange}
    />
)}
<ButtonContainer>
<CustomButton
    title={"Confirmar"}
    onPress={handleCallNext}
    ></CustomButton>
</ButtonContainer>
</ContentContainer>
</Container>
</>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import { fadeInAnim } from "../../../../../assets/animations";
import styled from "styled-components";

export const PatientButtonContainer = styled.div`
width: 100%;
display: grid;
grid-template-columns: 1fr 2fr 1fr;
align-items: start;
gap: 2rem;
`;

export const PatientContainer = styled.div`
grid-column: 2;
display: flex;
flex-direction: column;
align-items: center;
gap: 1rem;
width: 100%;
`;

export const NextButton = styled.button`
grid-column: 3;

```

```
justify-self: start;
display: flex;
font-size: 16px;
font-weight: 400;
align-items: center;
justify-content: center;
background-color: ${color.front};
border: 0.7px solid ${color.gray};
border-radius: 5px;
padding: 1rem;
gap: 1rem;
cursor: pointer;
transition: all 0.3s ease;
white-space: nowrap;
flex-shrink: 0;
margin-left: 1rem;

&:hover {
  background-color: ${color.primary};
  color: ${color.front};
  border-color: ${color.primary};
  transform: translateY(-1px);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

&:active {
  transform: scale(0.98);
  transition: all 0.1s ease;
}
';

export const ButtonContainer = styled.div`width: 100%;`;
;

export const ContentContainer = styled.div`display: flex;
flex-direction: column;
gap: 3rem;
width: fit-content;
`;
;

export const Container = styled.div`display: flex;
```

```

min-height: 100%;
flex-direction: column;
gap: 1rem;
align-items: center;

${fadeInAnim}
';

```

index.tsx

```

import { TextValue } from "../../../../../components/textValue";
import { FrontContainer } from "../../../../../components/frontContainer";
import { ManchesterTriage } from "@hisius/enums/src";
import { Select } from "../../../../../components/select";
import { Container, Title } from "./styles";
import type { IPatient } from "@hisius/interfaces/src";

interface AddCardProps {
  name: string;
  patient: IPatient;
  dateHourAttendance?: string;
  error?: string;
  selectedClassification: string;
  onClassificationChange: (classification: string) => void;
}

const formatDate = (dateString: string): string => {
  if (!dateString) return "Não informado";
  try {
    const date = new Date(dateString);
    return date.toLocaleDateString("pt-BR");
  } catch {
    return "Data inválida";
  }
};

const formatDateTime = (dateTimeString?: string): string => {
  if (!dateTimeString) return "Não informado";
  try {
    const date = new Date(dateTimeString);
    return date.toLocaleString("pt-BR");
  } catch {
    return "Data/hora inválida";
  }
};

```

```

};

export function AddCard({
  name,
  patient,
  dateHourAttendance,
  selectedClassification,
  onClassificationChange,
  error,
}: AddCardProps) {
  const classificationOptions = Object.values(ManchesterTriage).map(
    (value) => ({
      value: value,
      label: value,
    })
  );
}

const handleSelectChange = (event: React.ChangeEvent<HTMLSelectElement>) => {
  onClassificationChange(event.target.value);
};

const handleSelectClick = (event: React.MouseEvent) => {
  event.stopPropagation();
};

return (
  <FrontContainer>
    <Container>
      <Title>{name}</Title>

      <TextValue title="ID:">{patient.id}</TextValue>
      <TextValue title="Idade:">{patient.age} anos</TextValue>
      <TextValue title="Sexo:">{patient.gender}</TextValue>
      <TextValue title="Data de Nascimento:">
        {formatDate(patient.birthDate)}
      </TextValue>
      <TextValue title="Nome da M  e:">{patient.motherName}</TextValue>
      <TextValue title="N  mero CNS:">{patient.cnsNumber}</TextValue>

      {dateHourAttendance && (
        <TextValue title="Data/Hora Atendimento:">
          {formatDateTime(patient.dateHourAttendance)}
        </TextValue>
      )}
)
}

```

```

<Select
  label="Classificação"
  options={classificationOptions}
  onClick={handleSelectClick}
  onChange={handleSelectChange}
  value={selectedClassification}
  error={error}
/>
</Container>
</FrontContainer>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import styled from "styled-components";

export const Container = styled.div`
  display: flex;
  flex-direction: column;
  gap: 2rem;
  width: 100%;
`;

export const Title = styled.h1`
  text-align: center;
`;

export const ValuesContainer = styled.div`
  height: fit-content;
  padding: 1rem;
  display: flex;
  flex-direction: column;
  gap: 2rem;
  background-color: ${color.background};
`;

```

index.tsx

```

import { useState, useEffect } from "react";
import { QueueHeader } from "../../../../../components/navbar";

```

```

import Toggle from "../../components/toggle";
import { PatientCard } from "./components/patientCard";
import type { IPatient } from "@hisius/interfaces/src";
import Pagination from "../../../../../components/pagination";
import { FaPlus } from "react-icons/fa";
import { Sidebar } from "../../../../../components/sidebar";
import { useNavigate } from "react-router-dom";
import { Queue } from "@hisius/services";
import { useNotification } from "../../../../../components/notification/context";
import Popup from "../../../../../components/popup";
import CustomButton from "@hisius/ui/components/Button";
import CustomInput from "@hisius/ui/components/CustomInput";
import { LuDoorOpen } from "react-icons/lu";
import {
  Container,
  InputAndButtonContainer,
  NextButton,
  PatientButtonContainer,
  PatientContainer,
  PopupText,
} from "./styles";
import { usePageTitle } from "../../../../../hooks/PageTitle";

export function Employee() {
  const [searchTerm, setSearchTerm] = useState("");
  const [room, setRoom] = useState("");
  const [patients, setPatients] = useState<IPatient[]>([]);
  const [currentPage, setCurrentPage] = useState<number>(1);
  const [isTriage, setIsTriage] = useState<boolean>(false);
  const [nextPatient, setNextPatient] = useState<IPatient>();
  const [reloadFlag, setReloadFlag] = useState(false);
  const totalItems = patients.length || 0;
  const navigate = useNavigate();
  const { addNotification } = useNotification();
  const [isPopupOpen, setIsPopupOpen] = useState(false);
  const queueService = new Queue();

  usePageTitle("Funcionário - Hisius");

  const handleReloadFlag = () => {
    setReloadFlag((prev) => !prev);
  };

  const generateCard = (patients: IPatient[]) => {

```

```

return patients.map((patient) => (
  <PatientCard
    key={patient.id || 0}
    patient={patient}
    onChange={handleReloadFlag}
  />
));
};

useEffect(() => {
  const fetchPatients = async () => {
    try {
      const patientsData = await queueService.getPatientByQueue(
        isTriage,
        searchTerm
      );
      setPatients(patientsData.patients);
    } catch (error) {
      setPatients([]);
    }
  };
};

const timeoutId = setTimeout(() => {
  if (searchTerm.length >= 3 || searchTerm.length === 0) {
    fetchPatients();
  }
}, 500);

return () => clearTimeout(timeoutId);
}, [searchTerm, isTriage, isPopupOpen, reloadFlag]);

const handleToggleChange = async (checked: boolean) => {
  setIsTriage(checked);
};

const handleClick = async () => {
  const nextPatient = patients.find((patient) => !patient.classification);
  const nextPatientId = nextPatient?.id || patients[0]?.id;

  if (!nextPatientId) {
    addNotification("Nenhum paciente na fila", "warning");
    return;
  }
}

```

```

try {
  const patient = await queueService.getPatient(Number(nextPatientId));
  setNextPatient(patient);
  setIsPopupOpen(true);
} catch (err: any) {
  const errors = err.response?.data?.errors || [];
  const messages = errors
    .map((error: any) => error?.message || error)
    .filter(Boolean);

  if (messages.length === 0) {
    messages.push(
      err.response?.data?.message ||
      err.message ||
      "Erro ao chamar paciente"
    );
  }

  messages.forEach((message: string) => {
    addNotification(message, "error");
  });
}

const handleClickNext = async () => {
  try {
    const next = await queueService.getNextPatient(isTriage, room);

    if (!next?.id) {
      addNotification("Nenhum paciente na fila", "warning");
      return;
    }

    if (isTriage) navigate(`/funcionario/filas/${next.id}`);
    addNotification("Paciente chamado com sucesso", "success");
    setIsPopupOpen(false);
  } catch (err: any) {
    const errors = err.response?.data?.errors || [];
    const messages = errors
      .map((error: any) => error?.message || error)
      .filter(Boolean);

    if (messages.length === 0) {
      messages.push(
        err.response?.data?.message ||
        err.message ||
        "Erro ao chamar paciente"
      );
    }
  }
}

```

```

    err.response?.data?.message ||
    err.message ||
    "Erro ao chamar paciente"
);
}

messages.forEach((message: string) => {
  addNotification(message, "error");
});

};

const handlePageChange = (page: number) => {
  setCurrentPage(page);
  console.log(`Changed to page: ${page}`);
};

return (
  <>
  <Sidebar />
  <Popup
    isOpen={isPopupOpen}
    onClose={() => setIsPopupOpen(false)}
    title={`Chamar ${nextPatient?.name} para a próxima fila`}
    size="medium"
  >
    <PopupText>Para qual sala você deseja chamar o paciente?</PopupText>

    <InputAndButtonContainer>
      <CustomInput
        value={room}
        onChangeText={setRoom}
        placeholder="Digite o nome da sala"
        icon={<LuDoorOpen />}
      />
      <CustomButton title={"Chamar"} onPress={handleClickNext} />
    </InputAndButtonContainer>
  </Popup>
  <Container className="containerSide">
    <QueueHeader
      queueTitle="Consultar Paciente"
      queueSubtitle="Consulta paciente na fila"
      searchTerm={searchTerm}
      onChange={setSearchTerm}
    >

```

```

placeholder="Buscar pacientes...""
canSearch
/>
<Toggle
  labels={{ on: "Triagem", off: "Atendimento" }}
  onToggle={handleToggleChange}
/>
<PatientButtonContainer>
  <PatientContainer>
    {patients.length > 0 ? (
      generateCard(patients)
    ) : (
      <p>Nenhum paciente encontrado</p>
    )}
  </PatientContainer>
  <NextButton onClick={handleClick}>
    <FaPlus />
    Próximo paciente
  </NextButton>
</PatientButtonContainer>
<Pagination
  totalItems={totalItems}
  itemsPerPage={10}
  currentPage={currentPage}
  onPageChange={handlePageChange}
  maxVisibleButtons={3}
/>
</Container>
</>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import { fadeInAnim } from "../../../../../assets/animations";
import styled from "styled-components";

export const PatientButtonContainer = styled.div`
  width: 100%;
  display: grid;
  grid-template-columns: 1fr 2fr 1fr;
  align-items: start;
  gap: 2rem;

```

```
@media (max-width: 969px) {
  grid-template-columns: 1fr 1fr;
}

@media (max-width: 480px) {
  grid-template-columns: 1fr;
}
';

export const PatientContainer = styled.div`grid-column: 2; display: flex; flex-direction: column; align-items: center; gap: 1rem; width: 100%;`

@media (max-width: 969px) {
  grid-column: 1 / span 2;
  order: 2;
}

@media (max-width: 480px) {
  grid-column: 1;
}
';

export const NextButton = styled.button`grid-column: 3; justify-self: start; display: flex; font-size: 16px; font-weight: 400; align-items: center; justify-content: center; background-color: ${color.front}; border: 0.7px solid ${color.gray}; border-radius: 5px; padding: 1rem; gap: 1rem; cursor: pointer; transition: all 0.3s ease; white-space: nowrap;
```

```
flex-shrink: 0;
margin-left: 1rem;

@media (max-width: 969px) {
  grid-column: 1;
  order: 1;
  margin-left: 0;
}

@media (max-width: 480px) {
  grid-column: 1;
  justify-self: stretch;
  order: 1;
  margin-top: 1rem;
}

&:hover {
  background-color: ${color.primary};
  color: ${color.front};
  border-color: ${color.primary};
  transform: translateY(-1px);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

&:active {
  transform: scale(0.98);
  transition: all 0.1s ease;
}
';

export const InputAndButtonContainer = styled.div`  

  display: grid;  

  grid-template-columns: 2fr 1fr;  

  gap: 1rem;  

`;

export const PopupText = styled.p`  

  margin-bottom: 2rem;  

  width: 100%;  

  text-align: center;  

`;

export const Container = styled.div`
```

```
display: flex;
min-height: 100%;
flex-direction: column;
gap: 1rem;
align-items: center;
${fadeInAnim}
';
```

index.tsx

```
import type { IPatient } from "@hisius/interfaces/src";
import {
  Container,
  Description,
  NameTitle,
  SelectedCardContainer,
  TitleContainer,
  TriageBadge,
  PatientInfoGrid,
  InfoItem,
  InfoLabel,
  InfoValue,
} from "./styles";
import { useState } from "react";
import { TextValue } from "../../../../../components/textValue";
import { ManchesterTriage } from "@hisius/enums/src";
import { Select } from "../../../../../components/select";
import Button from "@hisius/ui/components/Button";
import { useNotification } from "../../../../../components/notification/context";
import { Queue } from "@hisius/services";
import { capitalizeWords } from "../../../../../utils";
import { useTruncate } from "../../../../../hooks/UseTruncate";

interface PatientCardProp {
  key: number;
  patient: IPatient;
  onChange: () => void;
}

export function PatientCard(props: PatientCardProp) {
  const [isSelected, setIsSelected] = useState(false);
  const [selectedClassification, setSelectedClassification] = useState<string>(
    props.patient.classification || ""
  );
}
```

```

const { addNotification } = useNotification();
const queueService = new Queue();

const { ref: nameRef, isTruncated } = useTruncate();

const classificationOptions = Object.values(ManchesterTriage).map(
  (value) => ({
    value,
    label: value,
  })
);

const handleCardClick = () => {
  setSelected(!isSelected);
};

const handleSelectClick = (event: React.MouseEvent) => {
  event.stopPropagation();
};

const handleSelectChange = (event: React.ChangeEvent<HTMLSelectElement>) => {
  setSelectedClassification(event.target.value);
};

const handleButtonClick = async () => {
  try {
    await queueService.updateClassification(
      Number(props.patient.id),
      selectedClassification
    );
    props.onChange?.();
    addNotification("Classificação alterada com sucesso", "success");
    setSelected(false);
  } catch (err: any) {
    const errors = err.response?.data?.errors || [];
    const messages = errors
      .map((error: any) => error?.message || error)
      .filter(Boolean);

    if (messages.length === 0) {
      messages.push(
        err.response?.data?.message ||
        err.message ||
        "Erro ao mudar classificação do paciente"
      );
    }
  }
};

```

```

    );
}

messages.forEach((message: string) => {
  addNotification(message, "error");
});

}
};

const formatDate = (dateString: string | undefined) => {
  if (!dateString) return "";
  const date = new Date(dateString);
  return date.toLocaleDateString("pt-BR");
};

const formatDateTime = (dateTimeString: string | undefined) => {
  if (!dateTimeString) return "";
  const date = new Date(dateTimeString);
  return date.toLocaleString("pt-BR");
};

const notSelectedCard = () => {
  return (
    <Description>
      {props.patient.classification ? (
        <>
          Classificação de risco:{' '}
          <TriageBadge type={props.patient.classification}>
            {capitalizeWords(props.patient.classification.toString())}
          </TriageBadge>
        </>
      ) : (
        "Paciente aguardando avaliação triagem"
      )}
    </Description>
  );
};

const selectedCard = () => {
  return (
    <SelectedCardContainer>
      <PatientInfoGrid>
        <InfoItem>
          <InfoLabel>ID:</InfoLabel>

```

```

<InfoValue>{props.patient.id}</InfoValue>
</InfoItem>
<InfoItem>
  <InfoLabel>Idade:</InfoLabel>
  <InfoValue>{props.patient.age} anos</InfoValue>
</InfoItem>
<InfoItem>
  <InfoLabel>Sexo:</InfoLabel>
  <InfoValue>{props.patient.gender}</InfoValue>
</InfoItem>
<InfoItem>
  <InfoLabel>Data de Nascimento:</InfoLabel>
  <InfoValue>{formatDate(props.patient.birthDate)}</InfoValue>
</InfoItem>
<InfoItem>
  <InfoLabel>Nome da Mãe:</InfoLabel>
  <InfoValue>{props.patient.motherName}</InfoValue>
</InfoItem>
<InfoItem>
  <InfoLabel>Número CNS:</InfoLabel>
  <InfoValue>{props.patient.cnsNumber}</InfoValue>
</InfoItem>
<InfoItem>
  <InfoLabel>Data/Hora Atendimento:</InfoLabel>
  <InfoValue>
    {formatDateTime(props.patient.dateHourAttendance)}
  </InfoValue>
</InfoItem>
</PatientInfoGrid>

{props.patient.classification ? (
  <>
    <TextValue title="Classificação:">
      <TriageBadge type={props.patient.classification}>
        {capitalizeWords(props.patient.classification.toString())}
      </TriageBadge>
    </TextValue>

    <Select
      label="Classificação"
      options={classificationOptions}
      onClick={handleSelectClick}
      onChange={handleSelectChange}
      value={selectedClassification}>

```

```

        />
        <Button title="Alterar classificação" onPress={handleButtonClick} />
      </>
    ) : (
      """
    )}
  </SelectedCardContainer>
);
};

return (
<>
<Container
  $isSelected={isSelected}
  type={props.patient.classification}
  onClick={handleCardClick}
>
  <TitleContainer>
    <NameTitle
      ref={nameRef}
      $isSelected={isSelected}
      title={isTruncated ? props.patient.name : undefined}
    >
      {props.patient.name}
    </NameTitle>
  </TitleContainer>
  {!isSelected ? notSelectedCard() : selectedCard()}
</Container>
</>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import styled, { css } from "styled-components";
import { ManchesterTriage } from "@hisius/enums";
import {

  contractAnimationAnim,
  expandAnimationAnim,
  tabAnim,
} from "../../../../../assets/animations";

interface TextProps {

```

```
    type?: ManchesterTriage;
}

interface MutableProps {
  $isSelected?: boolean;
}

const mapColors: Record<ManchesterTriage, string> = {
  [ManchesterTriage.Emergency]: color.triage.emergency,
  [ManchesterTriage.VeryUrgent]: color.triage.veryUrgent,
  [ManchesterTriage.Urgent]: color.triage.urgent,
  [ManchesterTriage.Standard]: color.triage.standard,
  [ManchesterTriage.NonUrgent]: color.triage.nonUrgent,
} as const;

const getTriageColor = (type?: ManchesterTriage): string => {
  if (!type) return color.triage.emergency;
  return mapColors[type] || color.triage.emergency;
};

export const PatientInfoGrid = styled.div`
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 8px;
  margin-top: 12px;
  border-radius: 4px;
`;

export const InfoItem = styled.div`
  display: flex;
  flex-direction: column;
  gap: 2px;
`;

export const InfoLabel = styled.span`
  font-size: 0.75rem;
  font-weight: 600;
  color: #6c757d;
  text-transform: uppercase;
`;

export const InfoValue = styled.span`
  font-size: 0.875rem;
  color: #495057;
`;
```

```
font-weight: 500;
';

export const Container = styled.div<TextProps & MutableProps>`  
width: 100%;  
min-height: 100px;  
background-color: ${color.front};  
border-radius: 5px;  
border-left: 2px solid ${(props) => getTriageColor(props.type)};  
display: flex;  
flex-direction: column;  
justify-content: center;  
padding: 1rem;  
position: relative;  
gap: 0.4rem;  
cursor: pointer;  
transform-origin: top center;  
transition: all 0.3s ease;  
${(props: MutableProps) =>  
  props.$isSelected ? expandAnimationAnim : contractAnimationAnim};  
  
&:hover {  
  transform: scale(1.005);  
  box-shadow: 0px 4px 5px rgba(0, 0, 0, 0.05);  
}  
  
z-index: 1;  
  
@media (max-width: 768px) {  
  min-height: 90px;  
  padding: 0.8rem;  
  gap: 0.3rem;  
}  
  
@media (max-width: 480px) {  
  min-height: 80px;  
  padding: 0.6rem;  
  gap: 0.2rem;  
  
  ${(props: MutableProps) =>  
    props.$isSelected ? `padding: 1rem 0.5rem;` : `padding: 0.6rem;`};  
  
&:hover {  
  transform: none;
```

```
        }
    }
';

export const SelectContainer = styled.div`  
  display: flex;  
  gap: 1rem;  
  align-items: center;  
  font-size: 13px;  
  font-weight: 400;  
  flex-wrap: wrap;  
  
  @media (max-width: 768px) {  
    gap: 0.8rem;  
    font-size: 12px;  
  }  
  
  @media (max-width: 480px) {  
    gap: 0.5rem;  
    font-size: 11px;  
    justify-content: space-between;  
  }  
';  
  
export const TriageBadge = styled.span<TextProps>'  
  color: ${props => getTriageColor(props.type)};  
  font-size: 13px;  
  font-weight: 400;  
  
  @media (max-width: 768px) {  
    font-size: 12px;  
  }  
  
  @media (max-width: 480px) {  
    font-size: 11px;  
  }  
';  
  
export const SelectedCardContainer = styled.div`  
  min-width: 40%;  
  display: flex;  
  flex-direction: column;  
  gap: 1.5rem;  
  padding: 2rem;
```

```
overflow: hidden;
transform-origin: top;

@media (max-width: 1024px) {
  min-width: 50%;
  padding: 1.5rem;
}

@media (max-width: 768px) {
  min-width: 60%;
  padding: 1rem;
  gap: 1rem;
}

@media (max-width: 480px) {
  min-width: 100%;
  padding: 0.8rem;
  gap: 0.8rem;
}
';

export const TitleContainer = styled.div`  

position: relative;  

width: 100%;  

height: 30px;  

  @media (max-width: 768px) {  

    height: 26px;  

  }  

  @media (max-width: 480px) {  

    height: 24px;  

  }  

';

export const NameTitle = styled.h1<MutableProps>`  

font-size: 16px;  

font-weight: 600;  

margin: 0;  

transition: all 0.3s ease;  

position: absolute;  

left: ${(props: MutableProps) => (props.$isSelected ? "50%" : "0")};  

transform: ${(props: MutableProps) =>  

  props.$isSelected ? "translateX(-50%)" : "translateX(0)"};
```

```

white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
max-width: 100%;
display: block;

${(props: MutableProps) =>
  !props.$isSelected &&
  css`  

    ${Container}:hover & {
      ${tabAnim}
    }
  `}
}

@media (max-width: 768px) {
  font-size: 15px;
  left: ${(props: MutableProps) => (props.$isSelected ? "50%" : "0")};
  transform: ${(props: MutableProps) =>
    props.$isSelected ? "translateX(-50%)" : "translateX(0)"};
}
}

@media (max-width: 480px) {
  font-size: 14px;
  position: static;
  transform: none;
  text-align: ${(props: MutableProps) =>
    props.$isSelected ? "center" : "left"};
  margin-bottom: 0.2rem;
}

${(props: MutableProps) =>
  !props.$isSelected &&
  css`  

    ${Container}:hover & {
      animation: none;
    }
  `}
};

export const Description = styled.p`  

  font-size: 13px;  

  font-weight: 200;  

  transition: all 0.3s ease;
}

```

```

line-height: 1.4;

${Container}:hover & {
  ${tabAnim}
  color: #555;
}

@media (max-width: 768px) {
  font-size: 12px;
  line-height: 1.3;
}

@media (max-width: 480px) {
  font-size: 11px;
  line-height: 1.2;

  ${Container}:hover & {
    animation: none;
  }
}

';

```

index.tsx

```

import { useState, useEffect } from "react";
import { QueueHeader } from "../../../../../components/navbar";
import Toggle from "../../../../../components/toggle";
import { PatientCard } from "./components/patientCard";
import type { IPatient } from "@hisius/interfaces/src";
import Pagination from "../../../../../components/pagination";
import { Sidebar } from "../../../../../components/sidebar";
import { Queue } from "@hisius/services";
import { Container, PatientButtonContainer, PatientContainer } from "./styles";
import { usePageTitle } from "../../../../../hooks/PageTitle";

export function InRoom() {
  const [searchTerm, setSearchTerm] = useState("");
  const [patients, setPatients] = useState<IPatient[]>([]);
  const [currentPage, setCurrentPage] = useState<number>(1);
  const [isTriage, setIsTriage] = useState<boolean>(false);
  const [reloadFlag, setReloadFlag] = useState(false);
  const totalItems = patients.length || 0;
  const queueService = new Queue();

```

```

usePageTitle("Salas - Hisius");

const handleReloadFlag = () => {
  setReloadFlag((prev) => !prev);
};

const generateCard = (patients: IPatient[]) => {
  return patients.map((patient) => (
    <PatientCard
      key={patient.id || 0}
      id={patient.id}
      isTriage={isTriage}
      patient={patient}
      onChange={handleReloadFlag}
    />
  ));
};

useEffect(() => {
  const fetchPatients = async () => {
    try {
      const patientsData = await queueService.getPatientInRoomByQueue(
        isTriage,
        searchTerm
      );
      setPatients(patientsData.patients);
    } catch (error) {
      setPatients([]);
    }
  };
};

const timeoutId = setTimeout(() => {
  if (searchTerm.length >= 3 || searchTerm.length === 0) {
    fetchPatients();
  }
}, 500);

return () => clearTimeout(timeoutId);
}, [searchTerm, isTriage, reloadFlag]);

const handleToggleChange = async (checked: boolean) => {
  setIsTriage(checked);
};

```

```

const handlePageChange = (page: number) => {
  setCurrentPage(page);
};

return (
  <>
  <Sidebar />
  <Container className="containerSide">
    <QueueHeader
      queueTitle="Consultar Salas"
      queueSubtitle="Consulta paciente nas salas"
      searchTerm={searchTerm}
      onChange={setSearchTerm}
      placeholder="Buscar pacientes..."
      canSearch
    />
    <Toggle
      labels={{ on: "Triagem", off: "Atendimento" }}
      onToggle={handleToggleChange}
    />
    <PatientButtonContainer>
      <PatientContainer>
        {patients.length > 0 ? (
          generateCard(patients)
        ) : (
          <p>Nenhum paciente encontrado</p>
        )}
      </PatientContainer>
    </PatientButtonContainer>
    <Pagination
      totalItems={totalItems}
      itemsPerPage={10}
      currentPage={currentPage}
      onPageChange={handlePageChange}
      maxVisibleButtons={3}
    />
  </Container>
</>
);
}

```

styles.ts

```
import { color } from "@hisius/ui/theme/colors";
```

```
import { fadeInAnim } from "../../../../../assets/animations";
import styled from "styled-components";

export const PatientButtonContainer = styled.div`  
  width: 100%;  
  display: grid;  
  grid-template-columns: 1fr 2fr 1fr;  
  align-items: start;  
  gap: 2rem;  
  
  @media (max-width: 969px) {  
    grid-template-columns: 1fr 1fr;  
  }  
  
  @media (max-width: 480px) {  
    grid-template-columns: 1fr;  
  }  
`;  
  
export const PatientContainer = styled.div`  
  grid-column: 2;  
  display: flex;  
  flex-direction: column;  
  align-items: center;  
  gap: 1rem;  
  width: 100%;  
  
  @media (max-width: 969px) {  
    grid-column: 1 / span 2;  
    order: 2;  
  }  
  
  @media (max-width: 480px) {  
    grid-column: 1;  
  }  
`;  
  
export const NextButton = styled.button`  
  grid-column: 3;  
  justify-self: start;  
  display: flex;  
  font-size: 16px;  
  font-weight: 400;  
  align-items: center;  
  justify-content: center;  
  background-color: ${color.front};  
  border: 0.7px solid ${color.gray};  
  border-radius: 5px;  
  padding: 1rem;  
  gap: 1rem;
```

```
cursor: pointer;
transition: all 0.3s ease;
white-space: nowrap;
flex-shrink: 0;
margin-left: 1rem;

@media (max-width: 969px) {
  grid-column: 1;
  order: 1;
  margin-left: 0;
}

@media (max-width: 480px) {
  grid-column: 1;
  justify-self: stretch;
  order: 1;
  margin-top: 1rem;
}

&:hover {
  background-color: ${color.primary};
  color: ${color.front};
  border-color: ${color.primary};
  transform: translateY(-1px);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.1);
}

&:active {
  transform: scale(0.98);
  transition: all 0.1s ease;
}

export const InputAndButtonContainer = styled.div`  

  display: grid;  

  grid-template-columns: 2fr 1fr;  

  gap: 1rem;  

`;

export const PopupText = styled.p`  

  margin-bottom: 2rem;  

  width: 100%;  

  text-align: center;  

`;

export const Container = styled.div`
```

```
display: flex;
min-height: 100%;
flex-direction: column;
gap: 1rem;
align-items: center;
${fadeInAnim}
';
```

index.tsx

```
import type { IPatient } from "@hisius/interfaces/src";
import {
  Container,
  Description,
  NameTitle,
  SelectedCardContainer,
  TitleContainer,
  TriageBadge,
  PatientInfoGrid,
  InfoItem,
  InfoLabel,
  InfoValue,
} from "./styles";
import { useState } from "react";
import { TextValue } from "../../../../../components/textValue";
import Button from "@hisius/ui/components/Button";
import { capitalizeWords } from "../../../../../utils";
import { useNavigate } from "react-router-dom";
import { Queue } from "@hisius/services/src";
import { useNotification } from "../../../../../components/notification/context";

interface PatientCardProp {
  key: number;
  patient: IPatient;
  id?: number;
  isTriage: boolean;
  onChange: () => void;
}

export function PatientCard(props: PatientCardProp) {
  const [isSelected, setIsSelected] = useState(false);
  const navigate = useNavigate();
  const queueService = new Queue();
  const { addNotification } = useNotification();
```

```

const handleCardClick = () => {
  setSelected(!isSelected);
};

const handleButtonClick = async () => {
  if (props.isTriage) navigate(`/funcionario/filas/${props.id}`);
  else {
    try {
      await queueService.finishTreatment(props.id!);
      addNotification("Atendimento finalizado com sucesso", "success");
    } catch (err: any) {
      const errors = err.response?.data?.errors || [];
      const messages = errors
        .map((error: any) => error?.message || error)
        .filter(Boolean);

      if (messages.length === 0) {
        messages.push(
          err.response?.data?.message ||
          err.message ||
          "Erro ao chamar finalizar cadastro"
        );
      }
    }
  }

  messages.forEach((message: string) => {
    addNotification(message, "error");
  });
};

props.onChange?.();
setSelected(false);
};

const formatDate = (dateString: string | undefined) => {
  if (!dateString) return "";
  const date = new Date(dateString);
  return date.toLocaleDateString("pt-BR");
};

const notSelectedCard = () => {
  return (
    <Description>
  )
};

```

```

{props.patient.classification ? (
  <>
    Classificação de risco:{" "}
    <TriageBadge type={props.patient.classification}>
      {capitalizeWords(props.patient.classification.toString())}
    </TriageBadge>
  </>
) : (
  "Paciente aguardando avaliação triagem"
)
</Description>
);
};

const selectedCard = () => {
  return (
    <SelectedCardContainer>
      <PatientInfoGrid>
        <InfoItem>
          <InfoLabel>ID:</InfoLabel>
          <InfoValue>{props.patient.id}</InfoValue>
        </InfoItem>
        <InfoItem>
          <InfoLabel>Idade:</InfoLabel>
          <InfoValue>{props.patient.age} anos</InfoValue>
        </InfoItem>
        <InfoItem>
          <InfoLabel>Sexo:</InfoLabel>
          <InfoValue>{props.patient.gender}</InfoValue>
        </InfoItem>
        <InfoItem>
          <InfoLabel>Data de Nascimento:</InfoLabel>
          <InfoValue>{formatDate(props.patient.birthDate)}</InfoValue>
        </InfoItem>
        <InfoItem>
          <InfoLabel>Nome da Mãe:</InfoLabel>
          <InfoValue>{props.patient.motherName}</InfoValue>
        </InfoItem>
        <InfoItem>
          <InfoLabel>Número CNS:</InfoLabel>
          <InfoValue>{props.patient.cnsNumber}</InfoValue>
        </InfoItem>
      </PatientInfoGrid>
    )
  );
};

```

```

{props.patient.classification ? (
  <>
    <TextValue title="Classificação:>
      <TriageBadge type={props.patient.classification}>
        {capitalizeWords(props.patient.classification.toString())}
      </TriageBadge>
    </TextValue>
  </>
) : (
  ""
)
}

<Button
  title={props.isTriage ? "Ver informações" : "Finalizar atendimento"}
  onPress={handleButtonClick}
/>
</SelectedCardContainer>
);
};

return (
  <>
  <Container
    $isSelected={isSelected}
    type={props.patient.classification}
    onClick={handleCardClick}
  >
    <TitleContainer>
      <NameTitle $isSelected={isSelected}>{props.patient.name}</NameTitle>
    </TitleContainer>
    {!isSelected ? notSelectedCard() : selectedCard()}
  </Container>
  </>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import styled, { css } from "styled-components";
import { ManchesterTriage } from "@hisius/enums";
import {
  contractAnimationAnim,
  expandAnimationAnim,
  tabAnim,

```

```
        } from "../../../../../assets/animations";

interface TextProps {
  type?: ManchesterTriage;
}

interface MutableProps {
  isSelected?: boolean;
}

const mapColors: Record<ManchesterTriage, string> = {
  [ManchesterTriage.Emergency]: color.triage.emergency,
  [ManchesterTriage.VeryUrgent]: color.triage.veryUrgent,
  [ManchesterTriage.Urgent]: color.triage.urgent,
  [ManchesterTriage.Standard]: color.triage.standard,
  [ManchesterTriage.NonUrgent]: color.triage.nonUrgent,
} as const;

const getTriageColor = (type?: ManchesterTriage): string => {
  if (!type) return color.triage.emergency;
  return mapColors[type] || color.triage.emergency;
};

export const PatientInfoGrid = styled.div`
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
  gap: 8px;
  margin-top: 12px;
  border-radius: 4px;
`;

export const InfoItem = styled.div`
  display: flex;
  flex-direction: column;
  gap: 2px;
`;

export const InfoLabel = styled.span`
  font-size: 0.75rem;
  font-weight: 600;
  color: #6c757d;
  text-transform: uppercase;
`;
```

```
export const InfoValue = styled.span`  
  font-size: 0.875rem;  
  color: #495057;  
  font-weight: 500;  
  `;  
  
export const Container = styled.div<TextProps & MutableProps>`  
  width: 100%;  
  min-height: 100px;  
  background-color: ${color.front};  
  border-radius: 5px;  
  border-left: 2px solid ${(props) => getTriageColor(props.type)};  
  display: flex;  
  flex-direction: column;  
  justify-content: center;  
  padding: 1rem;  
  position: relative;  
  gap: 0.4rem;  
  cursor: pointer;  
  transform-origin: top center;  
  transition: all 0.3s ease;  
  ${(props: MutableProps) =>  
    props.$isSelected ? expandAnimationAnim : contractAnimationAnim};  
  
  &:hover {  
    transform: scale(1.005);  
    box-shadow: 0px 4px 5px rgba(0, 0, 0, 0.05);  
  }  
  
  z-index: 1;  
  
  @media (max-width: 768px) {  
    min-height: 90px;  
    padding: 0.8rem;  
    gap: 0.3rem;  
  }  
  
  @media (max-width: 480px) {  
    min-height: 80px;  
    padding: 0.6rem;  
    gap: 0.2rem;  
  
    ${(props: MutableProps) =>  
      props.$isSelected ? `padding: 1rem 0.5rem;` : `padding: 0.6rem;`};  
  }
```

```
&:hover {
    transform: none;
}
};

export const SelectContainer = styled.div`  

display: flex;  

gap: 1rem;  

align-items: center;  

font-size: 13px;  

font-weight: 400;  

flex-wrap: wrap;  

@media (max-width: 768px) {  

    gap: 0.8rem;  

    font-size: 12px;  

}  

@media (max-width: 480px) {  

    gap: 0.5rem;  

    font-size: 11px;  

    justify-content: space-between;  

}  

`;

export const TriageBadge = styled.span<TextProps>`  

color: ${props => getTriageColor(props.type)};  

font-size: 13px;  

font-weight: 400;  

@media (max-width: 768px) {  

    font-size: 12px;  

}  

@media (max-width: 480px) {  

    font-size: 11px;  

}  

`;

export const SelectedCardContainer = styled.div`  

min-width: 40%;  

display: flex;
```

```
flex-direction: column;
gap: 1.5rem;
padding: 2rem;
overflow: hidden;
transform-origin: top;

@media (max-width: 1024px) {
  min-width: 50%;
  padding: 1.5rem;
}

@media (max-width: 768px) {
  min-width: 60%;
  padding: 1rem;
  gap: 1rem;
}

@media (max-width: 480px) {
  min-width: 100%;
  padding: 0.8rem;
  gap: 0.8rem;
}
';

export const TitleContainer = styled.div`  

  position: relative;  

  width: 100%;  

  height: 30px;  

  @media (max-width: 768px) {  

    height: 26px;  

  }  

  @media (max-width: 480px) {  

    height: 24px;  

  }  

';

export const NameTitle = styled.h1<MutableProps>`  

  font-size: 16px;  

  font-weight: 600;  

  margin: 0;  

  transition: all 0.3s ease;  

  position: absolute;
```

```

left: ${(props: MutableProps) => (props.$isSelected ? "50%" : "0")};
transform: ${(props: MutableProps) =>
  props.$isSelected ? "translateX(-50%)" : "translateX(0)"};

white-space: nowrap;
overflow: hidden;
text-overflow: ellipsis;
max-width: 100%;
display: block;

${(props: MutableProps) =>
  !props.$isSelected &&
  css`  

    ${Container}:hover & {
      ${tabAnim}
    }
  `}
}

@media (max-width: 768px) {
  font-size: 15px;
  left: ${(props: MutableProps) => (props.$isSelected ? "50%" : "0")};
  transform: ${(props: MutableProps) =>
    props.$isSelected ? "translateX(-50%)" : "translateX(0)"};
}

@media (max-width: 480px) {
  font-size: 14px;
  position: static;
  transform: none;
  text-align: ${(props: MutableProps) =>
    props.$isSelected ? "center" : "left"};
  margin-bottom: 0.2rem;

  ${(props: MutableProps) =>
    !props.$isSelected &&
    css`  

      ${Container}:hover & {
        animation: none;
      }
    `}
  };
}

export const Description = styled.p`
```

```

font-size: 13px;
font-weight: 200;
transition: all 0.3s ease;
line-height: 1.4;

${Container}:hover & {
  ${tabAnim}
  color: #555;
}

@media (max-width: 768px) {
  font-size: 12px;
  line-height: 1.3;
}

@media (max-width: 480px) {
  font-size: 11px;
  line-height: 1.2;

  ${Container}:hover & {
    animation: none;
  }
}
';

```

index.tsx

```

import { useState, useEffect } from "react";
import { QueueHeader } from "../../../../../components/navbar";
import { Sidebar as ManagerSidebar } from "../../../../../manager/components/sidebar";
import { Sidebar as EmployeeSidebar } from "../../../../../employee/components/sidebar";
import { useNotification } from "../../../../../components/notification/context";
import Popup from "../../../../../components/popup";
import CustomButton from "@hisius/ui/components/Button";
import CustomInput from "@hisius/ui/components/CustomInput";
import { HiOutlineEnvelope, HiOutlineLockClosed } from "react-icons/hi2";
import {
  Container,
  ProfileContainer,
  FormContainer,
  Section,
  SectionTitle,
  InputGroup,
  SecurityActions,

```

```

SecurityItem,
SecurityIcon,
SecurityTextContainer,
SecurityTitle,
SecurityDescription,
ActionsContainer,
PopupText,
StyledIcon,
} from "./styles";
import LocalStorageManager from "@hisius/services/src/helpers/localStorageManager";
import { Auth } from "@hisius/services/src";
import { color } from "@hisius/ui/theme/colors";
import { useFormErrors } from "../../../../../hooks/FormErrors";
import { useAuth } from "../../../../../context/authContext";
import { usePageTitle } from "../../../../../hooks/PageTitle";

export function ProfileScreen() {
  const { addNotification } = useNotification();
  const [isEmailPopupOpen, setIsEmailPopupOpen] = useState(false);
  const [newEmail, setNewEmail] = useState("");
  const [name, setName] = useState("");
  const [email, setEmail] = useState("");
  const [initialData, setInitialData] = useState({ name: "", email: "" });
  const [userRole, setUserRole] = useState<number | null>(null);
  usePageTitle("Perfil - Hisius");

  const AuthService = new Auth();
  const { errors, clearFieldError, handleApiErrors } = useFormErrors();
  const auth = useAuth();

  const loadProfileData = async () => {
    try {
      const user = LocalStorageManager.getUser();
      if (user && user.role !== undefined) {
        setUserRole(user.role);
      }
      const profileData = await AuthService.getProfile();
      setName(profileData.name || "");
      setEmail(profileData.email || "");
      setInitialData({
        name: profileData.name || "",
        email: profileData.email || "",
      });
      setNewEmail(profileData.email || "");
    }
  }
}

```

```

} catch (err: any) {
  console.error("Erro ao carregar perfil:", err);
  addNotification("Erro ao carregar dados do perfil", "error");
  if (err.response?.data) {
    handleApiErrors(err.response.data);
  }
}
};

useEffect(() => {
  loadProfileData();
}, []);
}

const handleChangePass = async () => {
  try {
    await AuthService.changePass(email);
    addNotification(
      "Código de verificação enviado para seu email!",
      "success"
    );
    setIsEmailPopupOpen(false);
  } catch (err: any) {
    console.error("Erro ao alterar senha:", err);
    addNotification(
      "Erro ao enviar código de verificação. Tente novamente.",
      "error"
    );
    if (err.response?.data) {
      handleApiErrors(err.response.data);
    }
  }
};

const handleLogout = () => {
  auth.logout();
  addNotification("Logout realizado com sucesso!", "success");
  window.location.reload();
};

const handleNewEmailChange = (value: string) => {
  setNewEmail(value);
  clearFieldError("email");
};

```

```

const handleNameChange = (value: string) => {
  setName(value);
  clearFieldError("name");
};

const handleSave = async () => {
  try {
    await AuthService.changeName(name);
    addNotification("Perfil atualizado com sucesso!", "success");
    setInitialData({ name, email });
  } catch (err: any) {
    addNotification(
      err.response.data.message || "Erro ao atualizar perfil",
      "error"
    );
  }
};

const handleEmailSubmit = async () => {
  try {
    await AuthService.changeEmail(newEmail);
    addNotification(
      "Código de verificação enviado para o novo email!",
      "success"
    );
    setIsEmailPopupOpen(false);
    clearFieldError("email");
  } catch (err: any) {
    console.error("Erro ao alterar email:", err);
    addNotification(
      "Erro ao enviar código de verificação. Tente novamente.",
      "error"
    );
    if (err.response?.data) {
      handleApiErrors(err.response.data);
    }
  }
};

const renderSidebar = () => {
  if (userRole === 0) {
    return <ManagerSidebar />;
  } else if (userRole === 2) {
    return <EmployeeSidebar />;
  }
};

```

```
        }

        return null;
    };

const hasValidChanges = () => name !== initialData.name;

return (
    <>
    {renderSidebar()}

<Popup
    isOpen={isEmailPopupOpen}
    onClose={() => setIsEmailPopupOpen(false)}
    title="Alterar Email"
    size="medium"
>
    <CustomInput
        placeholder="Digite o novo email"
        value={newEmail}
        onChangeText={handleNewEmailChange}
        error={errors.email}
        style={{ marginBottom: 30 }}
    />
    <PopupText>
        Enviaremos um email de verificação para confirmar que este email
        pertence a você.
    </PopupText>
    <CustomButton title="Alterar" onPress={handleEmailSubmit} />
</Popup>

<Container className="containerSide">
    <QueueHeader
        queueTitle="Seu Perfil"
        queueSubtitle="Gerencie suas informações pessoais"
    />

    <ProfileContainer>
        <FormContainer>
            <Section>
                <SectionTitle>Informações Pessoais</SectionTitle>
                <InputGroup>
                    <CustomInput
                        placeholder="Nome completo"
                        value={name}
                    />
                
```

```
onChangeText={handleNameChange}
error={errors.name}
/>
</InputGroup>
<InputGroup>
<CustomInput
placeholder="Email"
value={email}
onChangeText={() => {}}
disabled
/>
</InputGroup>
</Section>

<Section>
<SectionTitle>Segurança</SectionTitle>
<SecurityActions>
<SecurityItem onClick={() => setIsEmailPopupOpen(true)}>
<SecurityIcon>
<HiOutlineEnvelope size={24} />
</SecurityIcon>
<SecurityTextContainer>
<SecurityTitle>Alterar Email</SecurityTitle>
<SecurityDescription>
Modifique seu endereço de email
</SecurityDescription>
</SecurityTextContainer>
</SecurityItem>

<SecurityItem onClick={handleChangePass}>
<SecurityIcon>
<HiOutlineLockClosed size={24} />
</SecurityIcon>
<SecurityTextContainer>
<SecurityTitle>Alterar Senha</SecurityTitle>
<SecurityDescription>
Atualize sua senha de acesso
</SecurityDescription>
</SecurityTextContainer>
</SecurityItem>

<SecurityItem onClick={handleLogout}>
<SecurityIcon>
<StyledIcon size={24} />
```

```

        </SecurityIcon>
        <SecurityTextContainer>
            <SecurityTitle style={{ color: color.error.error }}>
                Sair da Conta
            </SecurityTitle>
            <SecurityDescription>Saia da sua conta</SecurityDescription>
        </SecurityTextContainer>
    </SecurityItem>
    </SecurityActions>
</Section>

<ActionsContainer>
    <CustomButton
        title="Salvar Alterações"
        onPress={handleSave}
        disabled={!isValidChanges()}
    />
</ActionsContainer>
</FormContainer>
</ProfileContainer>
</Container>
</>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import { HiOutlineArrowRightOnRectangle } from "react-icons/hi2";
import styled from "styled-components";

export const Container = styled.div`
    flex: 1;
    background-color: ${color.background};
`;

export const ProfileContainer = styled.div`
    flex: 1;
    background-color: ${color.background};
    padding: 24px;
`;

export const FormContainer = styled.div`
    display: flex;

```

```
flex-direction: column;
margin-top: 30px;
gap: 32px;
';

export const Section = styled.div`  

  display: flex;  

  flex-direction: column;  

  gap: 16px;  

'';

export const SectionTitle = styled.h2`  

  font-size: 18px;  

  font-weight: 600;  

  color: ${color.text};  

  margin-bottom: 8px;  

  padding-left: 12px;  

  border-left: 3px solid ${color.primary};  

'';

export const InputGroup = styled.div`  

  width: 100%;  

'';

export const SecurityActions = styled.div`  

  display: flex;  

  flex-direction: column;  

  gap: 12px;  

'';

export const SecurityItem = styled.button`  

  display: flex;  

  flex-direction: row;  

  align-items: center;  

  padding: 16px;  

  background-color: ${color.front};  

  border-radius: 12px;  

  border: 1px solid ${color.gray};  

  cursor: pointer;  

  width: 100%;  

  text-align: left;  

  &:hover {  

    background-color: ${color.gray};  

  }  

`;
```

```
}

';

export const SecurityIcon = styled.div`  
  margin-right: 12px;  
`;

export const SecurityTextContainer = styled.div`  
  flex: 1;  
`;

export const StyledIcon = styled(HiOutlineArrowRightOnRectangle)`  
  & path {  
    color: ${color.error.error};  
  }  
`;

export const SecurityTitle = styled.span`  
  font-size: 16px;  
  font-weight: 600;  
  color: ${color.text};  
  margin-bottom: 2px;  
  display: block;  
`;

export const SecurityDescription = styled.span`  
  font-size: 14px;  
  color: ${color.text};  
  display: block;  
`;

export const ActionsContainer = styled.div`  
  margin-top: 40px;  
  gap: 24px;  
`;

export const PopupText = styled.p`  
  font-size: 14px;  
  margin-bottom: 2rem;  
  text-align: center;  
`;
```

index.tsx

```
import React, { useState } from "react";
import {
  HiOutlineMail,
  HiOutlineLockClosed,
  HiOutlineUser,
  HiOutlineEye,
  HiOutlineEyeOff,
} from "react-icons/hi";
import {
  LoginContainer,
  LoginFormWrapper,
  Subtitle,
  Title,
  Link,
  FormsWrapper,
  FormContainer,
  TogglePanel,
  TogglePanelContent,
  ToggleTitle,
  ToggleText,
  ToggleButton,
  ToggleOverlay,
  StatusMessage,
  InfoMessage,
  ContactInfo,
  ContactItem,
} from "./style";

import CustomInput from "@hisius/ui/components/CustomInput";
import CustomButton from "@hisius/ui/components/Button";
import { Auth } from "@hisius/services/src";
import { useFormErrors } from "../../../../../hooks/FormErrors";
import { useAuth } from "../../../../../context/authContext";
import { usePageTitle } from "../../../../../hooks/PageTitle";

export interface LoginFormData {
  email: string;
  password: string;
}

export interface RegisterFormData {
  name: string;
  email: string;
```

```
password: string;
confirmPassword: string;
}

const LoginForm: React.FC = () => {
  const [isLogin, setIsLogin] = useState(true);
  const [loginData, setLoginData] = useState<LoginFormData>({
    email: "",
    password: "",
  });
  const [registerData, setRegisterData] = useState<RegisterFormData>({
    name: "",
    email: "",
    password: "",
    confirmPassword: "",
  });
  const [successMessage, setSuccessMessage] = useState("");
  const [errorMessage, setErrorMessage] = useState("");

  usePageTitle("Autenticação");

  const {
    errors,
    handleApiErrors,
    setFieldError,
    clearFieldError,
    clearAllErrors,
  } = useFormErrors();

  const authService = new Auth();
  const auth = useAuth();

  const validateLoginForm = (): boolean => {
    const newErrors: Partial<LoginFormData> = {};

    if (!loginData.email) newErrors.email = "Email é obrigatório";
    if (!loginData.password) newErrors.password = "Senha é obrigatória";

    Object.keys(newErrors).forEach((field) => {
      setFieldError(field, newErrors[field as keyof LoginFormData]!);
    });

    return Object.keys(newErrors).length === 0;
  };
}
```

```

const validateRegisterForm = (): boolean => {
  const newErrors: Partial<RegisterFormData> = {};

  if (!registerData.name) newErrors.name = "Nome é obrigatório";
  if (!registerData.email) newErrors.email = "Email é obrigatório";
  if (!registerData.password) newErrors.password = "Senha é obrigatória";
  if (!registerData.confirmPassword)
    newErrors.confirmPassword = "Confirme sua senha";
  if (registerData.password !== registerData.confirmPassword) {
    newErrors.confirmPassword = "Senhas não coincidem";
  }

  Object.keys(newErrors).forEach((field) => {
    setFieldError(field, newErrors[field as keyof RegisterFormData]!);
  });

  return Object.keys(newErrors).length === 0;
};

const handleLoginSubmit = async () => {
  setErrorMessage("");
  setSuccessMessage("");

  if (!validateLoginForm()) return;

  try {
    const data = await authService.Login(loginData);
    const { accessToken, ...rest } = data;

    if (rest.role != 0 && rest.role != 2) {
      setErrorMessage("Você não tem permissão para acessar");
      return;
    }

    auth.login(rest, accessToken!);
    window.location.reload();
  } catch (err: any) {
    if (err.response?.data) {
      handleApiErrors(err.response.data);
      setErrorMessage(err.response.data.message || "Erro ao fazer login");
    }
  }
};

```

```

const handleRegisterSubmit = async () => {
  setErrorMessage("");
  setSuccessMessage("");

  if (!validateRegisterForm()) return;

  try {
    await authService.adminRegister(registerData);
    setSuccessMessage("Cadastro realizado com sucesso!");
    setTimeout(() => window.location.reload(), 2000);
  } catch (err: any) {
    if (err.response?.data) {
      handleApiErrors(err.response.data);
      setErrorMessage(err.response.data.message || "Erro ao cadastrar");
    }
  }
};

const handleInputChange =
  (form: "login" | "register", field: string) => (value: string) => {
  if (form === "login") {
    setLoginData((prev) => ({ ...prev, [field]: value }));
  } else {
    setRegisterData((prev) => ({ ...prev, [field]: value }));
  }
  if (errors[field]) {
    clearFieldError(field);
  }
  if (errorMessage) {
    setErrorMessage("");
  }
};

const toggleForm = () => {
  setIsLogin(!isLogin);
  clearAllErrors();
  setErrorMessage("");
  setSuccessMessage("");
};

const iconStyle = { width: "20px", height: "20px" };

return (

```

```

<LoginContainer>
  <FormsWrapper>
    <FormContainer position="left" isActive={isLogin}>
      <LoginFormWrapper onSubmit={(e) => e.preventDefault()}>
        <Title>Entrar</Title>
        <Subtitle>Entre na sua conta</Subtitle>

        {successMessage && (
          <StatusMessage variant="success">{successMessage}</StatusMessage>
        )}

        {errorMessage && (
          <StatusMessage variant="error">{errorMessage}</StatusMessage>
        )}

      <CustomInput
        value={loginData.email}
        onChangeText={handleInputChange("login", "email")}
        placeholder="Email"
        error={errors.email}
        icon={<HiOutlineMail style={iconStyle} />}
      />

      <CustomInput
        value={loginData.password}
        onChangeText={handleInputChange("login", "password")}
        placeholder="Senha"
        error={errors.password}
        visibilityOff={<HiOutlineEyeOff style={iconStyle} />}
        visibilityOn={<HiOutlineEye style={iconStyle} />}
        icon={<HiOutlineLockClosed style={iconStyle} />}
        secureTextEntry
      />
      <p>
        <Link href="/senha/esqueci">Esqueci minha senha</Link>
      </p>
      <p>
        Não tem uma conta? <Link onClick={toggleForm}>Cadastre-se</Link>
      </p>
      <CustomButton title="Entrar" onPress={handleLoginSubmit} />
    </LoginFormWrapper>
  </FormContainer>

  <FormContainer position="right" isActive={!isLogin}>

```

```

<LoginFormWrapper onSubmit={handleRegisterSubmit}>
  <Title>Criar Conta de Administrador</Title>

  <Subtitle>
    Cadastre-se para acessar o painel administrativo
  </Subtitle>

  <InfoMessage>
    <strong>Para criar sua conta:</strong>
    <span>Entre em contato conosco</span>
    <ContactInfo>
      <ContactItem>email@empresa.com</ContactItem>
      <ContactItem>(11) 99999-9999</ContactItem>
    </ContactInfo>
  </InfoMessage>

  <p>
    Já tem uma conta? <Link onClick={toggleForm}>Entre</Link>
  </p>
</LoginFormWrapper>
</FormContainer>

<TogglePanel isLogin={isLogin}>
  <ToggleOverlay />
  <TogglePanelContent isVisible={isLogin}>
    <ToggleTitle>Olá!</ToggleTitle>
    <ToggleText>
      Faça parte!
    </ToggleText>
    <ToggleButton onClick={toggleForm}>Criar conta</ToggleButton>
  </TogglePanelContent>

  <TogglePanelContent isVisible={!isLogin}>
    <ToggleTitle>Bom te ver de volta!</ToggleTitle>
    <ToggleText>
      Para continuar, insira as informações da sua conta.
    </ToggleText>
    <ToggleButton onClick={toggleForm}>Entrar</ToggleButton>
  </TogglePanelContent>
</TogglePanel>
</FormsWrapper>
</LoginContainer>
);
};

```

```
export default LoginForm;
```

style.ts

```
import { color } from "@hisius/ui/theme/colors";
import styled, { css } from "styled-components";

export const LoginContainer = styled.div`  

  min-height: 100vh;  

  display: flex;  

  align-items: center;  

  justify-content: center;  

  background: ${color.background};  

  overflow: hidden;  

  position: relative;  
  

  @media (max-width: 768px) {  

    padding: 15px;  

  }  

`;  
  

export const FormsWrapper = styled.div`  

  position: relative;  

  width: 100%;  

  max-width: 100%;  

  height: 100vh;  

  background: ${color.front};  

  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);  

  overflow: hidden;  

`;  
  

export const FormContainer = styled.div<{  

  position: "left" | "right";  

  isActive: boolean;  

}>`  

  position: absolute;  

  top: 0;  

  width: 50%;  

  height: 100%;  

  padding: 50px;  

  display: flex;  

  flex-direction: column;  

  justify-content: center;  

  transition: all 0.6s ease-in-out;  

  z-index: 1;  
  

${(props) =>  

  props.position === "left" &&
```

```

css`  

  left: 0;  

  transform: translateX(0%);  

`}  

${(props) =>  

  props.position === "right" &&  

  css`  

    right: 0;  

    transform: translateX(0%);  

`}  

@media (max-width: 768px) {  

  width: 100%;  

  position: relative;  

  transform: none !important;  

  display: ${(props) => (props.isActive ? "flex" : "none")};  

  padding: 50px 60px;  

  left: auto !important;  

  right: auto !important;  

}  

';  

export const StatusMessage = styled.div<{  

  variant: "success" | "error" | "info";  

}>  

  padding: 16px;  

  border-radius: 12px;  

  text-align: center;  

  margin-bottom: 16px;  

  border: 1px solid;  

  line-height: 1.5;  

$({{ variant }}) => {  

  switch (variant) {  

    case "success":  

      return `  

        background: ${color.error.ok}1a;  

        color: ${color.error.ok};  

        border-color: ${color.error.ok}4d;  

`;  

    case "error":  

      return `  

        background: ${color.error.error}1a;  

        color: ${color.error.error};  

        border-color: ${color.error.error}4d;  

`;  

    case "info":  

      return `  

        background: ${color.primary}0d;  

        color: ${color.text};  

`;  

  }
}

```

```
    border-color: ${color.primary}26;
  `;
  default:
    return "";
}
```;
}

export const LoginFormWrapper = styled.form`
width: 100%;
display: flex;
flex-direction: column;
gap: 20px;
`;

export const Title = styled.h1`
font-size: 28px;
font-weight: 700;
text-align: center;
margin-bottom: 8px;
`;

export const InfoMessage = styled.div`
background: ${color.front};
border: 1px solid ${color.gray};
border-radius: 8px;
padding: 12px;
margin: 16px 0;
display: flex;
flex-direction: column;
gap: 4px;

span {
 font-size: 14px;
}
`;

export const ContactInfo = styled.div`
display: flex;
flex-direction: column;
gap: 8px;
margin: 16px 0;
`;

export const ContactItem = styled.div`
display: flex;
align-items: center;
gap: 8px;
font-size: 14px;
padding: 4px 0;
```

```
`;

export const Subtitle = styled.p`
 text-align: center;
 color: ${color.text}80;
 margin-bottom: 30px;
`;

export const Footer = styled.div`
 margin-top: 25px;
 text-align: center;
`;

export const Link = styled.a`
 color: ${color.primary};
 text-decoration: none;
 transition: color 0.3s ease;
 cursor: pointer;

 &:hover {
 color: ${color.secondary};
 text-decoration: underline;
 }
`;

export const SignupText = styled.span`
 display: block;
 margin-top: 20px;
 color: ${color.text}80;
`;

export const TogglePanel = styled.div<{ isLogin: boolean }>`
 position: absolute;
 top: 0;
 left: ${({props}) => (props.isLogin ? "50%" : "0")};
 width: 50%;
 height: 100%;
 overflow: hidden;
 transition: all 0.6s ease-in-out;
 z-index: 2;

 @media (max-width: 768px) {
 display: none;
 }
`;

export const ToggleOverlay = styled.div`
 background: ${color.primary};
 color: white;
 height: 100%;
```

```

width: 100%;
display: flex;
position: relative;
`;

export const TogglePanelContent = styled.div` isVisible: boolean }>`
position: absolute;
top: 0;
left: 0;
width: 100%;
height: 100%;
display: flex;
flex-direction: column;
align-items: center;
justify-content: center;
padding: 40px;
text-align: center;
transition: all 0.3s ease-in-out;
opacity: ${(props) => (props.isVisible ? 1 : 0)};
`;

export const ToggleTitle = styled.h2`
font-size: 24px;
font-weight: 600;
margin-bottom: 20px;
color: white;
`;

export const ToggleText = styled.p`
margin-bottom: 30px;
color: rgba(255, 255, 255, 0.9);
line-height: 1.5;
`;

export const ToggleButton = styled.button`
background: transparent;
border: 2px solid white;
color: white;
width: 50%;
padding: 12px 30px;
border-radius: 15px;
font-size: 16px;
font-weight: 600;
cursor: pointer;
transition: all 0.3s ease;
`;

```

## index.tsx

```
import React, { useState, useEffect } from "react";
```

```

import { HiOutlineCheckCircle } from "react-icons/hi";
import CustomButton from "@hisius/ui/components/Button";
import { Auth } from "@hisius/services/src";
import { color } from "@hisius/ui/theme/colors";
import { useNavigate, useSearchParams } from "react-router-dom";
import {
 ScreenContainer,
 AuthCard,
 AuthForm,
 AuthHeader,
 ScreenTitle,
 ScreenDescription,
 StatusMessage,
 SuccessScreen,
} from "./style";
import { usePageTitle } from "../../hooks/PageTitle";

const ConfirmEmailScreen: React.FC = () => {
 const [isLoading, setIsLoading] = useState(false);
 const [error, setError] = useState("");
 const [isVerified, setIsVerified] = useState(false);

 const navigate = useNavigate();
 const [searchParams] = useSearchParams();
 const encodedToken = searchParams.get("token");
 const token = encodedToken ? decodeURIComponent(encodedToken) : null;

 const authService = new Auth();
 usePageTitle("Confirmar Email")

 useEffect(() => {
 if (!token) {
 navigate("/login");
 return;
 }
 handleVerifyEmail(token);
 }, [token]);

 const handleVerifyEmail = async (token: string): Promise<void> => {
 setIsLoading(true);
 setError("");
 try {
 await authService.resetEmail(token);
 }
 }
}

```

```

 setIsVerified(true);
 } catch (err: any) {
 setError(err.response.data.message || "Link expirado ou inválido.");
 } finally {
 setIsLoading(false);
 }
};

if (isVerified) {
 return (
 <ScreenContainer>
 <AuthCard>
 <SuccessScreen>
 <HiOutlineCheckCircle size={48} color={color.error.ok} />
 <AuthHeader>
 <ScreenTitle>Email Verificado!</ScreenTitle>
 <ScreenDescription>
 Seu email foi verificado com sucesso.
 </ScreenDescription>
 </AuthHeader>
 <CustomButton
 title="Ir para o Login"
 onPress={() => navigate("/login")}
 />
 </SuccessScreen>
 </AuthCard>
 </ScreenContainer>
);
}

return (
 <ScreenContainer>
 <AuthCard>
 <AuthForm onSubmit={(e) => e.preventDefault()} noValidate>
 <AuthHeader>
 <ScreenTitle>
 {error ? "Falha na Verificação" : "Verificar Email"}
 </ScreenTitle>
 <ScreenDescription>
 {error
 ? "Não foi possível verificar seu email."
 : "Estamos verificando seu email..."}
 </ScreenDescription>
 </AuthHeader>

```

```

{error && <StatusMessage variant="error">{error}</StatusMessage>}

{!error && isLoading && (
 <StatusMessage variant="info">Verificando...</StatusMessage>
)}
</AuthForm>
</AuthCard>
</ScreenContainer>
);
};

export default ConfirmEmailScreen;

```

## style.ts

```

import styled from "styled-components";
import { color } from "@hisius/ui/theme/colors";

export const ScreenContainer = styled.main`
 min-height: 100vh;
 display: flex;
 align-items: center;
 justify-content: center;
 background: ${color.background};
 padding: 16px;

 @media (max-width: 768px) {
 padding: 12px;
 }
`;

export const AuthCard = styled.article`
 position: relative;
 width: 100%;
 max-width: 500px;
 background: ${color.front};
 box-shadow: 0 10px 30px rgba(0, 0, 0, 0.15);
 border-radius: 20px;
 overflow: hidden;
 padding: 48px;

 @media (max-width: 768px) {
 padding: 32px 24px;
 }
`;

```

```
 max-width: 100%;
}
;

export const AuthForm = styled.form`
 width: 100%;
 display: flex;
 flex-direction: column;
 gap: 24px;
`;

export const AuthHeader = styled.header`
 text-align: center;
 margin-bottom: 16px;
`;

export const ScreenTitle = styled.h1`
 font-size: 28px;
 font-weight: 700;
 color: ${color.text};
 margin-bottom: 8px;
`;

export const ScreenDescription = styled.p`
 color: ${color.text}80;
 line-height: 1.6;
 margin-bottom: 24px;
`;

export const StatusMessage = styled.div<{
 variant: "success" | "error" | "info";
}>`
 padding: 16px;
 border-radius: 12px;
 text-align: center;
 margin-bottom: 16px;
 border: 1px solid;
 line-height: 1.5;

 ${({ variant }) => {
 switch (variant) {
 case "success":
 return `
 background: ${color.error.ok}1a;
```

```

 color: ${color.error.ok};
 border-color: ${color.error.ok}4d;
 `;
 case "error":
 return `
 background: ${color.error.error}1a;
 color: ${color.error.error};
 border-color: ${color.error.error}4d;
 `;
 case "info":
 return `
 background: ${color.primary}0d;
 color: ${color.text};
 border-color: ${color.primary}26;
 `;
 default:
 return "";
}
}
`;
}

export const SuccessScreen = styled.section`
 text-align: center;
 padding: 24px 0;
`;

export const EmailHighlight = styled.strong`
 color: ${color.primary};
 font-weight: 600;
`;

```

## types.ts

```

export interface ResetPasswordFormData {
 email: string;
 token: string;
 newPassword: string;
 confirmPassword: string;
}

export type ResetPasswordStep = "EMAIL_INPUT" | "EMAIL_SENT" |
"PASSWORD_RESET";

export interface FormErrors {
 email?: string;
}

```

```

 newPassword?: string;
 confirmPassword?: string;
}

```

## index.tsx

```

import React, { useState, useEffect } from "react";
import {
 HiOutlineLockClosed,
 HiOutlineCheckCircle,
 HiOutlineUser,
 HiOutlineMail,
} from "react-icons/hi";
import CustomInput from "@hisius/ui/components/CustomInput";
import CustomButton from "@hisius/ui/components/Button";
import { color } from "@hisius/ui/theme/colors";
import { useNavigate, useSearchParams } from "react-router-dom";
import {
 ScreenContainer,
 AuthCard,
 AuthForm,
 AuthHeader,
 ScreenTitle,
 ScreenDescription,
 StatusMessage,
 SuccessScreen,
 ScreenLink,
 ScreenFooter,
} from "./style";
import { Auth } from "@hisius/services/src";
import { useFormErrors } from "../../../../../hooks/FormErrors";
import { usePageTitle } from "../../../../../hooks/PageTitle";

interface FormData {
 name: string;
 email: string;
 password: string;
 confirmPassword: string;
}

const EmployeeRegistrationScreen: React.FC = () => {
 const [formData, setFormData] = useState<FormData>({
 name: "",
 email: "",
 });

```

```

password: "",
confirmPassword: "",
});

const [error, setError] = useState("");
const [isLoading, setIsLoading] = useState(false);
const [isSuccess, setIsSuccess] = useState(false);

const navigate = useNavigate();
const [searchParams] = useSearchParams();
const encodedToken = searchParams.get("token");
const token = encodedToken ? decodeURIComponent(encodedToken) : null;
const authService = new Auth();
const { errors, clearFieldError, handleApiErrors } = useFormErrors();
usePageTitle("Registrar Funcionário");

useEffect(() => {
 if (!token) {
 navigate("/login");
 }
}, [token, navigate]);

const handleSubmit = async (): Promise<void> => {
 setIsLoading(true);
 setError("");

 try {
 await authService.employeeRegister(formData, token!);
 setIsSuccess(true);
 setTimeout(() => navigate("/login"), 2000);
 } catch (err: any) {
 setError(err.response?.data?.message || "Erro ao completar cadastro.");
 if (err.response?.data) {
 handleApiErrors(err.response.data);
 }
 } finally {
 setIsLoading(false);
 }
};

const handleInputChange =
 (field: keyof FormData) =>
 (value: string): void => {
 setFormData((prev) => ({ ...prev, [field]: value }));
 clearFieldError(field);
 }
;
```

```

 if (error) setError("");
};

if (isSuccess) {
 return (
 <ScreenContainer>
 <AuthCard>
 <SuccessScreen>
 <HiOutlineCheckCircle size={48} color={color.error.ok} />
 <AuthHeader>
 <ScreenTitle>Cadastro Concluído!</ScreenTitle>
 <ScreenDescription>
 Sua conta foi criada com sucesso.

 Redirecionando para o login...
 </ScreenDescription>
 </AuthHeader>
 </SuccessScreen>
 </AuthCard>
 </ScreenContainer>
);
}

return (
 <ScreenContainer>
 <AuthCard>
 <AuthForm onSubmit={(e) => e.preventDefault()} noValidate>
 <AuthHeader>
 <ScreenTitle>Completar Cadastro</ScreenTitle>
 <ScreenDescription>
 Complete suas informações para criar sua conta
 </ScreenDescription>
 </AuthHeader>

 {error && <StatusMessage variant="error">{error}</StatusMessage>}

 <CustomInput
 value={formData.name}
 onChangeText={handleInputChange("name")}
 placeholder="Digite seu nome completo"
 error={errors.name}
 icon={<HiOutlineUser />}
 />
 </AuthForm>
 </AuthCard>
 </ScreenContainer>
);

```

```
<CustomInput
 value={formData.email}
 onChangeText={handleInputChange("email")}
 placeholder="Digite seu email"
 error={errors.email}
 icon={<HiOutlineMail />}
/>

<CustomInput
 value={formData.password}
 onChangeText={handleInputChange("password")}
 placeholder="Crie uma senha segura"
 error={errors.password}
 icon={<HiOutlineLockClosed />}
 secureTextEntry
/>

<CustomInput
 value={formData.confirmPassword}
 onChangeText={handleInputChange("confirmPassword")}
 placeholder="Confirme sua senha"
 error={errors.confirmPassword}
 icon={<HiOutlineLockClosed />}
 secureTextEntry
/>

<CustomButton
 title="Completar Cadastro"
 onPress={handleSubmit}
 disabled={isLoading}
/>

<ScreenFooter>
 Já tem uma conta?{" "}
 <ScreenLink onClick={() => navigate("/login")}>
 Fazer login
 </ScreenLink>
</ScreenFooter>
</AuthForm>
</AuthCard>
</ScreenContainer>
);
};
```

```
export default EmployeeRegistrationScreen;
```

## style.ts

```
import styled from "styled-components";
import { color } from "@hisius/ui/theme/colors";
```

```
export const ScreenContainer = styled.main`
 min-height: 100vh;
 display: flex;
 align-items: center;
 justify-content: center;
 background: ${color.background};
 padding: 16px;
```

```
 @media (max-width: 768px) {
 padding: 12px;
 }
`;
```

```
export const AuthCard = styled.article`
 position: relative;
 width: 100%;
 max-width: 500px;
 background: ${color.front};
 box-shadow: 0 10px 30px rgba(0, 0, 0, 0.15);
 border-radius: 20px;
 overflow: hidden;
 padding: 48px;
```

```
 @media (max-width: 768px) {
 padding: 32px 24px;
 max-width: 100%;
 }
`;
```

```
export const AuthForm = styled.form`
 width: 100%;
 display: flex;
 flex-direction: column;
 gap: 24px;
`;
```

```
export const AuthHeader = styled.header`
```

```
text-align: center;
margin-bottom: 16px;
';

export const ScreenTitle = styled.h1`

 font-size: 28px;

 font-weight: 700;

 color: ${color.text};

 margin-bottom: 8px;

`;

export const ScreenDescription = styled.p`

 color: ${color.text}80;

 line-height: 1.6;

 margin-bottom: 24px;

`;

export const StatusMessage = styled.div<{

 variant: "success" | "error" | "info";

}>`

 padding: 16px;

 border-radius: 12px;

 text-align: center;

 margin-bottom: 16px;

 border: 1px solid;

 line-height: 1.5;

${({ variant }) => {

 switch (variant) {

 case "success":

 return `

 background: ${color.error.ok}1a;

 color: ${color.error.ok};

 border-color: ${color.error.ok}4d;

 `;

 case "error":

 return `

 background: ${color.error.error}1a;

 color: ${color.error.error};

 border-color: ${color.error.error}4d;

 `;

 case "info":

 return `

 background: ${color.primary}0d;

 `;

 }
}
```

```
 color: ${color.text};
 border-color: ${color.primary}26;
 `;
 default:
 return "";
 }
}

`;

export const SuccessScreen = styled.section`

 text-align: center;

 padding: 24px 0;

`;

export const EmailHighlight = styled.strong`

 color: ${color.primary};

 font-weight: 600;

`;

export const ScreenLink = styled.button`

 color: ${color.primary};

 background: none;

 border: none;

 text-decoration: none;

 transition: all 0.3s ease;

 cursor: pointer;

 font-size: inherit;

 font-family: inherit;

 padding: 0;

 &:hover {

 color: ${color.secondary};

 text-decoration: underline;

 }

 &:focus-visible {

 outline: 2px solid ${color.primary};

 outline-offset: 2px;

 border-radius: 4px;

 }

 &:disabled {

 opacity: 0.6;

 cursor: not-allowed;

}
```

```

 }
 ';

export const ScreenFooter = styled.footer`
 margin-top: 24px;
 text-align: center;
 color: ${color.text}80;
`;

```

## types.ts

```

export interface ResetPasswordFormData {
 email: string;
 token: string;
 newPassword: string;
 confirmPassword: string;
}

export type ResetPasswordStep = "EMAIL_INPUT" | "EMAIL_SENT" |
"PASSWORD_RESET";

export interface FormErrors {
 email?: string;
 newPassword?: string;
 confirmPassword?: string;
}

```

## index.tsx

```

import React, { useState } from "react";
import { HiOutlineMail, HiOutlineCheckCircle } from "react-icons/hi";
import CustomInput from "@hisius/ui/components/CustomInput";
import CustomButton from "@hisius/ui/components/Button";
import { Auth } from "@hisius/services/src";
import { color } from "@hisius/ui/theme/colors";
import { useNavigate } from "react-router-dom";
import {
 ScreenContainer,
 AuthCard,
 AuthForm,
 AuthHeader,
 ScreenTitle,
 ScreenDescription,
}

```

```
>StatusMessage,
SuccessScreen,
EmailHighlight,
ScreenLink,
ScreenFooter,
} from "./style";
import type { ResetPasswordStep } from "./types";
import { useFormErrors } from "../../../../../hooks/FormErrors";
import { usePageTitle } from "../../../../../hooks/PageTitle";
const ResetPasswordScreen: React.FC = () => {
 const [currentStep, setCurrentStep] =
 useState<ResetPasswordStep>("EMAIL_INPUT");
 const [email, setEmail] = useState("");
 const [isLoading, setIsLoading] = useState(false);
 const [successMessage, setSuccessMessage] = useState("");
 const [errorMessage, setErrorMessage] = useState("");

 const authService = new Auth();
 const navigate = useNavigate();
 const { errors, clearFieldError, handleApiErrors } = useFormErrors();
 usePageTitle("Esqueci minha senha");

 const handleRequestResetLink = async (): Promise<void> => {
 setIsLoading(true);
 setErrorMessage("");

 try {
 await authService.changePass(email);

 setCurrentStep("EMAIL_SENT");
 setSuccessMessage(
 "Link de redefinição enviado para seu email com sucesso!"
);
 } catch (err: any) {
 setErrorMessage(
 err.response.data.message ||
 "Erro ao solicitar o link de redefinição. Tente novamente."
);
 if (err.response?.data) {
 handleApiErrors(err.response.data);
 }
 } finally {
 setIsLoading(false);
 }
 };
};
```

```

};

const handleInputChange = (value: string): void => {
 setEmail(value);
 clearFieldError("email");

 if (errorMessage) {
 setErrorMessage("");
 }
};

const handleResendLink = (): void => {
 setErrorMessage("");
 handleRequestResetLink();
};

const handleBackToLogin = (): void => {
 navigate("/login");
};

const handleSubmitEditing = (): void => {
 if (currentStep === "EMAIL_INPUT") {
 handleRequestResetLink();
 }
};

const renderEmailInputStep = () => (
 <>
 <AuthHeader>
 <ScreenTitle>Redefinir Senha</ScreenTitle>
 <ScreenDescription>
 Digite seu email para receber um link seguro de redefinição de senha
 </ScreenDescription>
 </AuthHeader>

 <StatusMessage variant="info">
 Link seguro

 Enviaremos um link de redefinição que expira em 1 hora
 </StatusMessage>

 <CustomInput
 value={email}
 onChangeText={handleInputChange}

```

```
placeholder="Seu endereço de email"
error={errors.email}
icon={<HiOutlineMail />}
onSubmitEditing={handleSubmitEditing}
returnKeyType="go"
/>

<CustomButton
 title="Enviar Link de Redefinição"
 onPress={handleRequestResetLink}
 disabled={isLoading}
/>
</>
);

const renderEmailSentStep = () => (
<SuccessScreen>
 <HiOutlineCheckCircle
 size={48}
 color={color.error.ok}
 style={{ marginBottom: "16px" }}
 />

 <AuthHeader>
 <ScreenTitle>Email Enviado!</ScreenTitle>
 <ScreenDescription>
 Enviamos um link de redefinição para:

 <EmailHighlight>{email}</EmailHighlight>
 </ScreenDescription>
 </AuthHeader>

 <StatusMessage variant="info">
 Verifique sua caixa de entrada

 Clique no link que enviamos para criar uma nova senha.

O link expira em 1 hora.
 </StatusMessage>

 <ScreenDescription>
 Não recebeu o email?{" "}
 <ScreenLink
 onClick={handleResendLink}
 type="button"
 >
```

```

 disabled={isLoading}
 >
 Reenviar link
 </ScreenLink>
 </ScreenDescription>
</SuccessScreen>
);

return (
<ScreenContainer>
 <AuthCard>
 <AuthForm
 onSubmit={(e) => {
 e.preventDefault();
 if (currentStep === "EMAIL_INPUT") {
 handleRequestResetLink();
 }
 }}
 noValidate
 >
 {successMessage && (
 <StatusMessage variant="success">{successMessage}</StatusMessage>
)}
 {errorMessage && (
 <StatusMessage variant="error">{errorMessage}</StatusMessage>
)}
 {currentStep === "EMAIL_INPUT" && renderEmailInputStep()}
 {currentStep === "EMAIL_SENT" && renderEmailSentStep()}

 <ScreenFooter>
 Lembrou sua senha? {" "}
 <ScreenLink onClick={handleBackToLogin}>
 Voltar para o login
 </ScreenLink>
 </ScreenFooter>
 </AuthForm>
</AuthCard>
</ScreenContainer>
);
};

export default ResetPasswordScreen;

```

## style.ts

```
import styled from "styled-components";
import { color } from "@hisius/ui/theme/colors";

export const ScreenContainer = styled.main`
 min-height: 100vh;
 display: flex;
 align-items: center;
 justify-content: center;
 background: ${color.background};
 padding: 16px;

 @media (max-width: 768px) {
 padding: 12px;
 }
`;

export const AuthCard = styled.article`
 position: relative;
 width: 100%;
 max-width: 500px;
 background: ${color.front};
 box-shadow: 0 10px 30px rgba(0, 0, 0, 0.15);
 border-radius: 20px;
 overflow: hidden;
 padding: 48px;

 @media (max-width: 768px) {
 padding: 32px 24px;
 max-width: 100%;
 }
`;

export const AuthForm = styled.form`
 width: 100%;
 display: flex;
 flex-direction: column;
 gap: 24px;
`;

export const AuthHeader = styled.header`
 text-align: center;
```

```
margin-bottom: 16px;
';

export const ScreenTitle = styled.h1`
 font-size: 28px;
 font-weight: 700;
 color: ${color.text};
 margin-bottom: 8px;
`;

export const ScreenDescription = styled.p`
 color: ${color.text}80;
 line-height: 1.6;
 margin-bottom: 24px;
`;

export const StatusMessage = styled.div<{
 variant: "success" | "error" | "info";
}>`
 padding: 16px;
 border-radius: 12px;
 text-align: center;
 margin-bottom: 16px;
 border: 1px solid;
 line-height: 1.5;

${({ variant }) => {
 switch (variant) {
 case "success":
 return `
 background: ${color.error.ok}1a;
 color: ${color.error.ok};
 border-color: ${color.error.ok}4d;
 `;
 case "error":
 return `
 background: ${color.error.error}1a;
 color: ${color.error.error};
 border-color: ${color.error.error}4d;
 `;
 case "info":
 return `
 background: ${color.primary}0d;
 color: ${color.text};
 `;
 }
}}
```

```
 border-color: ${color.primary}26;
 `;
}

default:
 return "";
}

```
`export const SuccessScreen = styled.section`  
text-align: center;  
padding: 24px 0;  
`;  
  
`export const EmailHighlight = styled.strong`  
color: ${color.primary};  
font-weight: 600;  
`;  
  
`export const ScreenLink = styled.button`  
color: ${color.primary};  
background: none;  
border: none;  
text-decoration: none;  
transition: all 0.3s ease;  
cursor: pointer;  
font-size: inherit;  
font-family: inherit;  
padding: 0;  
  
&:hover {  
  color: ${color.secondary};  
  text-decoration: underline;  
}  
  
&:focus-visible {  
  outline: 2px solid ${color.primary};  
  outline-offset: 2px;  
  border-radius: 4px;  
}  
  
&:disabled {  
  opacity: 0.6;  
  cursor: not-allowed;  
}
```

```
`;
```

```
export const ScreenFooter = styled.footer`
```

```
  margin-top: 24px;
```

```
  text-align: center;
```

```
  color: ${color.text}80;
```

```
`;
```

types.ts

```
export interface ResetPasswordFormData {
```

```
  email: string;
```

```
  token: string;
```

```
  newPassword: string;
```

```
  confirmPassword: string;
```

```
}
```

```
export type ResetPasswordStep = "EMAIL_INPUT" | "EMAIL_SENT";
```

```
export interface FormErrors {
```

```
  email?: string;
```

```
  newPassword?: string;
```

```
  confirmPassword?: string;
```

```
}
```

index.tsx

```
import React, { useState, useEffect } from "react";
import { HiOutlineLockClosed, HiOutlineCheckCircle } from "react-icons/hi";
import CustomInput from "@hisius/ui/components/CustomInput";
import CustomButton from "@hisius/ui/components/Button";
import { color } from "@hisius/ui/theme/colors";
import { useNavigate, useSearchParams } from "react-router-dom";
import {
  ScreenContainer,
  AuthCard,
  AuthForm,
  AuthHeader,
  ScreenTitle,
  ScreenDescription,
  StatusMessage,
  SuccessScreen,
  ScreenLink,
}
```

```

ScreenFooter,
} from "./style";
import { Auth } from "@hisius/services/src";
import { useFormErrors } from "../../../../../hooks/FormErrors";
import { usePageTitle } from "../../../../../hooks/PageTitle";

interface FormData {
  password: string;
  confirmPassword: string;
}

const PasswordSetupScreen: React.FC = () => {
  const [formData, setFormData] = useState<FormData>({
    password: "",
    confirmPassword: "",
  });
  const [error, setError] = useState("");
  const [isLoading, setIsLoading] = useState(false);
  const [isSuccess, setIsSuccess] = useState(false);

  const navigate = useNavigate();
  const [searchParams] = useSearchParams();
  const encodedToken = searchParams.get("token");
  const token = encodedToken ? decodeURIComponent(encodedToken) : null;
  const authService = new Auth();
  const { errors, clearFieldError, handleApiErrors } = useFormErrors();
  usePageTitle("Redefinir senha");
  useEffect(() => {
    if (!token) {
      navigate("/login");
    }
  }, [token, navigate]);

  const handleSubmit = async (): Promise<void> => {
    setIsLoading(true);
    setError("");

    try {
      await authService.resetPass(
        token!,
        formData.password,
        formData.confirmPassword
      );
      setIsSuccess(true);
    
```

```
    setTimeout(() => navigate("/login"), 2000);
} catch (err: any) {
  setError(err.response.data.message || "Erro ao redefinir senha.");
  if (err.response?.data) {
    handleApiErrors(err.response.data);
  }
} finally {
  setIsLoading(false);
}
};

const handleInputChange =
  (field: keyof FormData) =>
  (value: string): void => {
  setFormData((prev) => ({ ...prev, [field]: value }));
  clearFieldError(field);
  if (error) setError("");
};

if (isSuccess) {
  return (
    <ScreenContainer>
      <AuthCard>
        <SuccessScreen>
          <HiOutlineCheckCircle size={48} color={color.error.ok} />
          <AuthHeader>
            <ScreenTitle>Senha Redefinida!</ScreenTitle>
            <ScreenDescription>
              Sua senha foi redefinida com sucesso.
              <br />
              Redirecionando para o login...
            </ScreenDescription>
          </AuthHeader>
        </SuccessScreen>
      </AuthCard>
    </ScreenContainer>
  );
}

return (
  <ScreenContainer>
    <AuthCard>
      <AuthForm onSubmit={(e) => e.preventDefault()} noValidate>
        <AuthHeader>
```

```
<ScreenTitle>Redefinir Senha</ScreenTitle>
<ScreenDescription>
  Crie uma nova senha segura para sua conta
</ScreenDescription>
</AuthHeader>

{error && <StatusMessage variant="error">{error}</StatusMessage>}

<CustomInput
  value={formData.password}
  onChangeText={handleInputChange("password")}
  placeholder="Digite sua nova senha"
  error={errors.password}
  icon={<HiOutlineLockClosed />}
  secureTextEntry
/>

<CustomInput
  value={formData.confirmPassword}
  onChangeText={handleInputChange("confirmPassword")}
  placeholder="Confirme sua senha"
  error={errors.confirmPassword}
  icon={<HiOutlineLockClosed />}
  secureTextEntry
/>

<CustomButton
  title="Redefinir Senha"
  onPress={handleSubmit}
  disabled={isLoading}
/>

<ScreenFooter>
  Lembrou sua senha? {" "}
  <ScreenLink onClick={() => navigate("/login")}>
    Voltar para o login
  </ScreenLink>
</ScreenFooter>
</AuthForm>
</AuthCard>
</ScreenContainer>
);

};
```

```
export default PasswordSetupScreen;
```

style.ts

```
import styled from "styled-components";
import { color } from "@hisius/ui/theme/colors";
```

```
export const ScreenContainer = styled.main`  
  min-height: 100vh;  
  display: flex;  
  align-items: center;  
  justify-content: center;  
  background: ${color.background};  
  padding: 16px;
```

```
  @media (max-width: 768px) {  
    padding: 12px;  
  }  
`;
```

```
export const AuthCard = styled.article`  
  position: relative;  
  width: 100%;  
  max-width: 500px;  
  background: ${color.front};  
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.15);  
  border-radius: 20px;  
  overflow: hidden;  
  padding: 48px;
```

```
  @media (max-width: 768px) {  
    padding: 32px 24px;  
    max-width: 100%;  
  }  
`;
```

```
export const AuthForm = styled.form`  
  width: 100%;  
  display: flex;  
  flex-direction: column;  
  gap: 24px;  
`;
```

```
export const AuthHeader = styled.header`
```

```
text-align: center;
margin-bottom: 16px;
';

export const ScreenTitle = styled.h1`  

  font-size: 28px;  

  font-weight: 700;  

  color: ${color.text};  

  margin-bottom: 8px;  

`;

export const ScreenDescription = styled.p`  

  color: ${color.text}80;  

  line-height: 1.6;  

  margin-bottom: 24px;  

`;

export const StatusMessage = styled.div<{  

  variant: "success" | "error" | "info";  

}>`  

  padding: 16px;  

  border-radius: 12px;  

  text-align: center;  

  margin-bottom: 16px;  

  border: 1px solid;  

  line-height: 1.5;  

${({ variant }) => {  

  switch (variant) {  

    case "success":  

      return `  

        background: ${color.error.ok}1a;  

        color: ${color.error.ok};  

        border-color: ${color.error.ok}4d;  

      `;  

    case "error":  

      return `  

        background: ${color.error.error}1a;  

        color: ${color.error.error};  

        border-color: ${color.error.error}4d;  

      `;  

    case "info":  

      return `  

        background: ${color.primary}0d;  

      `;  

  }
}
```

```
    color: ${color.text};
    border-color: ${color.primary}26;
  ';
}

default:
  return "";
}

};

';

export const SuccessScreen = styled.section`  

  text-align: center;  

  padding: 24px 0;  

`;

export const EmailHighlight = styled.strong`  

  color: ${color.primary};  

  font-weight: 600;  

`;

export const ScreenLink = styled.button`  

  color: ${color.primary};  

  background: none;  

  border: none;  

  text-decoration: none;  

  transition: all 0.3s ease;  

  cursor: pointer;  

  font-size: inherit;  

  font-family: inherit;  

  padding: 0;  

  &:hover {  

    color: ${color.secondary};  

    text-decoration: underline;  

  }  

  &:focus-visible {  

    outline: 2px solid ${color.primary};  

    outline-offset: 2px;  

    border-radius: 4px;  

  }  

  &:disabled {  

    opacity: 0.6;  

    cursor: not-allowed;  

}
```

```

        }
   ';

export const ScreenFooter = styled.footer`
    margin-top: 24px;
    text-align: center;
    color: ${color.text}80;
`;

```

types.ts

```

export interface ResetPasswordFormData {
    email: string;
    token: string;
    newPassword: string;
    confirmPassword: string;
}

export type ResetPasswordStep = "EMAIL_INPUT" | "EMAIL_SENT" |
"PASSWORD_RESET";

export interface FormErrors {
    email?: string;
    newPassword?: string;
    confirmPassword?: string;
}

```

index.tsx

```

import { HiOutlineClock, HiOutlineHome } from "react-icons/hi2";
import { HiOutlineUserGroup } from "react-icons/hi2";
import { HiOutlineUser } from "react-icons/hi2";
import { TbGraph } from "react-icons/tb";
import { HiOutlineInformationCircle } from "react-icons/hi2";
import SidebarComponent from "../../../../../components/sidebar";

export function Sidebar() {
    const menuItems = [
        {
            icon: <HiOutlineHome />,
            text: "Principal",
            path: "/admin",
            exact: true,

```

```

},
{
  icon: <TbGraph />,
  text: "Relatório",
  path: "/admin/relatorio",
  exact: true,
},
{
  icon: <HiOutlineUserGroup />,
  text: "Funcionários",
  path: "/admin/funcionarios",
  exact: true,
},
{
  icon: <HiOutlineClock />,
  text: "Atividades",
  path: "/admin/logs",
  exact: true,
},
{
  icon: <HiOutlineInformationCircle />,
  text: "Administradores",
  path: "/admin/admins",
  exact: true,
},
{
  icon: <HiOutlineUser />,
  text: "Conta",
  path: "/perfil",
  exact: true,
},
];
}

return (
  <>
  <SidebarComponent menuItems={menuItems} />
</>
);
}

```

styles.ts

index.tsx

```

import { QueueHeader } from "../../../../../components/navbar";
import { Sidebar } from "../../../../../components/sidebar";
import { AdminCard } from "./components/adminCard";
import {
  AdminsContainer,
  Container,
  NoAdminsMessage,
  ContactContainer,
  DataContainer,
  ButtonContainer,
} from "./styles";
import { useEffect, useState } from "react";
import { Admin } from "@hisius/services";
import { useNotification } from "../../../../../components/notification/context";
import type { User } from "@hisius/interfaces";
import Pagination from "../../../../../components/pagination";
import Popup from "../../../../../components/popup";
import { HiOutlineEnvelope } from "react-icons/hi2";
import CustomButton from "@hisius/ui/components/Button";
import { usePageTitle } from "../../../../../hooks/PageTitle";

export function AdminsList() {
  const adminService = new Admin();
  const [searchTerm, setSearchTerm] = useState("");
  const [debouncedSearchTerm, setDebouncedSearchTerm] = useState("");
  const [admins, setAdmins] = useState<User[]>([]);
  const [selectedAdmin, setSelectedAdmin] = useState<User | null>(null);
  const [isPopupOpen, setIsPopupOpen] = useState(false);
  const [isLoading, setIsLoading] = useState(false);
  const [currentPage, setCurrentPage] = useState(1);
  const [totalItems, setTotalItems] = useState(0);
  const [itemsPerPage] = useState(12);
  const { addNotification } = useNotification();

  usePageTitle("Administradores - Hisius");

  const fetchAdmins = async () => {
    try {
      const apiPage = currentPage - 1;

      const searchName =
        debouncedSearchTerm.length >= 3 ? debouncedSearchTerm : undefined;

      const adminsData = await adminService.getAdmins(
        searchName,
        apiPage,
        itemsPerPage
      );
    }
  }
}

```

```
setAdmins(adminsData.users);
adminsData.pagination
  ? setTotalItems(adminsData.pagination.totalItems)
  : setTotalItems(adminsData.users.length);
} catch (error) {
  addNotification("Erro ao buscar administradores", "error");
  setAdmins([]);
  setTotalItems(0);
}
};

const handleAdminClick = (admin: User) => {
  setSelectedAdmin(admin);
  setIsPopupOpen(true);
};

const handleClosePopup = () => {
  setIsPopupOpen(false);
  setSelectedAdmin(null);
};

const handleMakeEmployee = async () => {
  if (!selectedAdmin) return;

  setIsLoading(true);
  try {
    await adminService.changeUserRole(selectedAdmin.id, 2);
    addNotification(
      "Administrador transformado em funcionário com sucesso!",
      "success"
    );
  }

  await fetchAdmins();

  handleClosePopup();
} catch (error) {
  addNotification(
    "Erro ao transformar administrador em funcionário",
    "error"
  );
} finally {
  setIsLoading(false);
}
};

const handlePageChange = (page: number) => {
  setCurrentPage(page);
  window.scrollTo({ top: 0, behavior: "smooth" });
};
```

```

useEffect(() => {
  const timer = setTimeout(() => {
    setDebouncedSearchTerm(searchTerm);
    setCurrentPage(1);
  }, 500);

  return () => clearTimeout(timer);
}, [searchTerm]);

useEffect(() => {
  fetchAdmins();
}, [currentPage, debouncedSearchTerm]);

return (
<>
  <Sidebar />
  <Popup
    isOpen={isPopupOpen}
    onClose={handleClosePopup}
    title={selectedAdmin?.name || "Administrador"}
    size="medium"
  >
    <ContactContainer>
      <p>Contato:</p>
      <DataContainer>
        <HiOutlineEnvelope /> {selectedAdmin?.email}
      </DataContainer>
      <ButtonContainer>
        <CustomButton
          title={isLoading ? "Processando..." : "Tornar Funcionário"}
          onPress={handleMakeEmployee}
          disabled={isLoading}
        />
      </ButtonContainer>
    </ContactContainer>
  </Popup>
  <Container className="containerSide">
    <QueueHeader
      queueTitle="Administradores"
      queueSubtitle="Administradores cadastrados"
      searchTerm={searchTerm}
      onChange={setSearchTerm}
      canSearch
      placeholder="Pesquisar administradores"
    />
    <AdminsContainer>
      {admins.length > 0 ? (
        admins.map((admin) => (
          <AdminCard

```

```

        key={admin.id}
        id={admin.id}
        name={admin.name}
        email={admin.email}
        onClick={() => handleAdminClick(admin)}
      />
    ))
):(
  <NoAdminsMessage>nenhum administrador encontrado</NoAdminsMessage>
)
</AdminsContainer>

{totalItems > 0 && (
  <Pagination
    totalItems={totalItems}
    itemsPerPage={itemsPerPage}
    currentPage={currentPage}
    onPageChange={handlePageChange}
    maxVisibleButtons={5}
  />
)
}>
</Container>
</>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import { fadeInAnim, subtleSlideAnim } from "../../../../../assets/animations";
import styled from "styled-components";

export const Container = styled.div`
  display: flex;
  position: relative;
  min-height: 100%;
  flex-direction: column;
  gap: 1rem;
  ${fadeInAnim}
`;

export const ButtonContainer = styled.div`
  display: flex;
  margin-top: 2rem;
  justify-content: center;
  width: 100%;
`;

```

```

export const DataContainer = styled.div`  

  display: flex;  

  gap: 1rem;  

  margin-top: 1rem;  
  

  & svg {  

    width: 20px;  

    height: 20px;  

  }  

`;  
  

export const ContactContainer = styled.div`  

  width: fit-content;  

  margin: 2rem auto;  

`;  
  

export const AdminsContainer = styled.div`  

  display: flex;  

  flex-direction: column;  

  width: 100%;  

  gap: 1rem;  

  ${subtleSlideAnim}  

`;  

export const NoAdminsMessage = styled.div`  

  display: flex;  

  justify-content: center;  

  align-items: center;  

  height: 200px;  

  color: ${color.text};  

  font-size: 16px;  

  width: 100%;  

  text-align: center;  

  grid-column: 1 / -1;  

`;

```

index.tsx

```

import { Container, Description, NameTitle, TitleContainer } from "./styles";  
  

interface AdminCardProp {  

  email: string;  

  onClick: () => void;  

  name: string;
}

```

```

    id: number;
}

export function AdminCard(props: AdminCardProp) {
  return (
    <>
    <Container onClick={props.onClick}>
      <TitleContainer>
        <NameTitle>{props.name}</NameTitle>
      </TitleContainer>
      <Description>ID: {props.id}</Description>

      <Description>{props.email}</Description>
    </Container>
    </>
  );
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import { clickAnim } from "../../../../../assets/animations";
import styled from "styled-components";

export const Container = styled.div`
  width: 100%;
  max-width: 729px;
  min-height: 100px;
  background-color: ${color.front};
  border-radius: 5px;
  border-left: 2px solid;
  border-color: ${color.primary};
  display: flex;
  flex-direction: column;
  justify-content: center;
  padding: 22px 20px;
  position: relative;
  gap: 0.4rem;
  transform-origin: top center;
  transition: all 0.3s ease;
  z-index: 1;

  cursor: pointer;

```

```
&:hover {
    transform: scale(1.005);
    box-shadow: 0px 4px 5px rgba(0, 0, 0, 0.05);
}
z-index: 1;

${clickAnim}
';

export const SelectContainer = styled.div`
    display: flex;
    gap: 1rem;
    align-items: center;
    font-size: 13px;
    font-weight: 400;
`;

export const SelectedCardContainer = styled.div`
    min-width: 40%;
    display: flex;
    flex-direction: column;
    gap: 1.5rem;
    padding: 2rem;
    overflow: hidden;
    transform-origin: top;
`;

export const TitleContainer = styled.div`
    position: relative;
    width: 100%;
    height: 30px;
`;

export const NameTitle = styled.h1`
    font-size: 16px;
    font-weight: 600;
    margin: 0;
    transition: all 0.3s ease;
    position: absolute;
`;

export const Description = styled.p`
    font-size: 13px;
    font-weight: 300;
`;
```

```
transition: all 0.3s ease;
';
```

index.tsx

```
import { QueueHeader } from "../../../../../components/navbar";
import { Sidebar } from "../../../../../components/sidebar";
import { HiOutlineClipboard } from "react-icons/hi2";
import {
  Container,
  InfoCardContainer,
  InfoContainer,
  InfoIcon,
  LogContainer,
  QueueContainer,
  SectionTitle,
  SubtitleInfo,
  TextContainer,
  TitleInfo,
} from "./styles";
import { Log } from "./components/log";
import { useEffect, useState } from "react";
import { Admin, Queue } from "@hisius/services";
import { useNotification } from "../../../../../components/notification/context";
import type { LogData } from "@hisius/interfaces/src";
import { usePageTitle } from "../../../../../hooks/PageTitle";

export function Dashboard() {
  const [hospitalCode, setHospitalCode] = useState<string>("");
  const [triageCount, setTriageCount] = useState<number>();
  const [treatmentCount, setTreatmentCount] = useState<number>();
  const [logs, setLogs] = useState<LogData[]>([]);
  const [loading, setLoading] = useState<boolean>(true);
  const adminService = new Admin();
  const queueService = new Queue();
  const { addNotification } = useNotification();

  usePageTitle("Administrador - Hisius");

  const formatDate = (dateString: string) => {
    const date = new Date(dateString);
    return date.toLocaleString("pt-BR");
  };
}
```

```

const handleCopyLog = (logId: number) => {
  navigator.clipboard.writeText(`Log ID: ${logId}`);
  addNotification("Log copiado para a área de transferência", "success");
};

useEffect(() => {
  const fetchPatientData = async () => {
    try {
      const hospitalInfo = await adminService.getHospitalInfo();
      const triageCount = await queueService.getQueueCount("triage");
      const treatmentCount = await queueService.getQueueCount("treatment");
      setTriageCount(triageCount);
      setTreatmentCount(treatmentCount);
      setHospitalCode(hospitalInfo.hospitalCode);
    } catch (error) {
      addNotification("Erro ao buscar informações", "error");
    }
  };
  fetchPatientData();
}, []);

useEffect(() => {
  const fetchLogs = async () => {
    try {
      const logsResponse = await adminService.getLogs(0, 8);
      setLogs(logsResponse.logs);
    } catch (error) {
      addNotification("Erro ao buscar logs", "error");
    } finally {
      setLoading(false);
    }
  };
  fetchLogs();
}, []);

return (
  <>
  <Sidebar />
  <Container className="containerSide">
    <QueueHeader
      queueTitle="Painel de operações"
      queueSubtitle="Verifique dados importantes"
)

```

```

/>
<InfoCardContainer>
  <QueueContainer>
    <InfoContainer>
      <TextContainer>
        <TitleInfo>Pessoas na Triagem</TitleInfo>
        <SubtitleInfo>{triageCount}</SubtitleInfo>
      </TextContainer>
      <InfoIcon>
        <HiOutlineClipboard />
      </InfoIcon>
    </InfoContainer>
    <InfoContainer>
      <TextContainer>
        <TitleInfo>Pessoas no Atendimento</TitleInfo>
        <SubtitleInfo>{treatmentCount}</SubtitleInfo>
      </TextContainer>
      <InfoIcon>
        <HiOutlineClipboard />
      </InfoIcon>
    </InfoContainer>
  </QueueContainer>

<InfoContainer style={{ justifySelf: "end" }}>
  <TextContainer>
    <TitleInfo>Codigo do Hospital</TitleInfo>
    <SubtitleInfo>{hospitalCode}</SubtitleInfo>
  </TextContainer>
  </InfoContainer>
</InfoCardContainer>
<SectionTitle>Atividades recentes</SectionTitle>
<LogContainer>
  {loading ? (
    <p>Carregando logs...</p>
  ) : logs.length > 0 ? (
    logs.map((log) => (
      <Log
        key={log.id}
        module={log.module}
        id={log.id.toString()}
        action={`\$${log.action} - ${formatDate(log.createdAt)}`}
        onClick={() => handleCopyLog(log.id)}
      />
    )))

```

```

) : (
  <p>Nenhum log encontrado</p>
)
</LogContainer>
</Container>
</>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import { fadeInAnim, subtleSlideAnim } from "../../../../../assets/animations";
import styled from "styled-components";

export const Container = styled.div`
  display: flex;
  position: relative;
  min-height: 100%;
  flex-direction: column;
  gap: 1rem;
  ${fadeInAnim}
`;

export const TitleInfo = styled.div`
  font-size: clamp(13px, 2vw, 25px);
  font-weight: 200;
`;

export const SubtitleInfo = styled.div`
  font-size: clamp(15px, 7vw, 25px);
  font-weight: 300;
  color: ${color.primary};
`;

export const TextContainer = styled.div`
  display: flex;
  flex-direction: column;
  justify-content: center;
  align-items: center;
  gap: 1rem;
`;

export const InfoIcon = styled.div`

```

```
& svg {
  width: 50px;
  height: 50px;
  color: ${color.text};
  transition: color 0.2s ease;
}

;

export const LogContainer = styled.div`  

  display: flex;  

  flex-direction: column;  

  width: 100%;  

  gap: 1rem;
`;  
  
export const QueueContainer = styled.div`  

  display: flex;  

  gap: 1rem;
`;  
  
@media (max-width: 754px) {  

  display: flex;  

  flex-direction: column;
}  
;  
  
export const InfoCardContainer = styled.div`  

  display: grid;  

  grid-template-columns: 1fr auto;  

  gap: 1rem;  

  width: 100%;  

  justify-content: space-between;
`;  
  
@media (max-width: 897px) {  

  display: flex;  

  flex-direction: column;  

  gap: 1rem;
}  
;  
  
export const SectionTitle = styled.h1`  

  margin-top: 2rem;  

  font-size: 24px;  

  font-weight: 400;  

  ${subtleSlideAnim}
`;
```

```

`;

export const InfoContainer = styled.div`
  display: flex;
  gap: 1rem;
  justify-content: space-around;
  align-items: center;
  padding: 1.5rem;

  background: ${color.front};
  border: 0.7px solid rgba(13, 19, 41, 0.12);
  border-radius: 5px;
  transition: all 0.2s ease;

  @media (max-width: 897px) {
    width: 100%;
  }
`;

```

index.tsx

```

import { CopyButton } from "../../../../../components/copyButton";
import { CodeContainer, Container, Text } from "./styles";

interface LogProps {
  onClick?: () => void;
  action: string;
  module: string;
  id: string;
}

export function Log({ onClick, module, action, id }: LogProps) {
  return (
    <Container>
      <Text>{module}</Text>
      <Text>{action}</Text>
      <CodeContainer>
        <Text>{id}</Text>
        <CopyButton onClick={onClick} />
      </CodeContainer>
    </Container>
  );
}

```

styles.ts

```
import { color } from "@hisius/ui/theme/colors";
import styled from "styled-components";

export const Container = styled.div`
  display: grid;
  width: 100%;
  gap: 1rem;
  grid-template-columns: 1fr 3fr 1fr;
  padding: 1rem;
  background: ${color.front};
  border: 0.7px solid rgba(13, 19, 41, 0.12);
  border-radius: 5px;

  & > *:nth-child(1) {
    border-right: 0.7px solid rgba(13, 19, 41, 0.12);
  }
  & * {
    display: flex;
    align-items: center;
  }
`;

export const CodeContainer = styled.div`
  display: flex;
  gap: 1rem;
`;

export const Text = styled.span`
  font-size: 13px;
  font-weight: 300;
  height: 100%;
`;
```

index.tsx

```
import { QueueHeader } from "../../../../../components/navbar";
import { Sidebar } from "../../components/sidebar";
import { Employee } from "./components/employee";
import { HiPlus, HiOutlineEnvelope } from "react-icons/hi2";
import {
  AddButton,
  ContactContainer,
```

```

Container,
CopyTextContainer,
DataContainer,
EmployeContainer,
TextToCopy,
NoEmployeesMessage,
ButtonContainer,
} from "./styles";
import { useEffect, useState } from "react";
import { Admin } from "@hisius/services";
import { useNotification } from "../../../../../components/notification/context";
import type { User } from "@hisius/interfaces";
import Popup from "../../../../../components/popup";
import { CopyButton } from "../../../../../components/copyButton";
import { copyToClipboard } from "../../../../../utils";
import Pagination from "../../../../../components/pagination";
import CustomButton from "@hisius/ui/components/Button";
import { usePageTitle } from "../../../../../hooks/PageTitle";

export function EmployeesList() {
  const adminService = new Admin();
  const [searchTerm, setSearchTerm] = useState("");
  const [debouncedSearchTerm, setDebouncedSearchTerm] = useState("");
  const [employees, setEmployees] = useState<User[]>([]);
  const [selectedEmployee, setSelectedEmployee] = useState<User | null>(null);
  const [isPopupOpen, setIsPopupOpen] = useState(false);
  const [isPopupCopyOpen, setIsPopupCopyOpen] = useState(false);
  const [code, setCode] = useState<string>("");
  const { addNotification } = useNotification();

  const [currentPage, setCurrentPage] = useState(1);
  const [totalItems, setTotalItems] = useState(0);
  const [itemsPerPage] = useState(12);
  const [isLoading, setIsLoading] = useState(false);

  usePageTitle("Funcionários - Hisius");

  const fetchEmployees = async () => {
    try {
      const apiPage = currentPage - 1;

      const searchName =
        debouncedSearchTerm.length >= 3 ? debouncedSearchTerm : undefined;
    }
  }
}

```

```
const employeesData = await adminService.getEmployees(
  searchName,
  apiPage,
  itemsPerPage
);

setEmployees(employeesData.users);
employeesData.pagination
  ? setTotalItems(employeesData.pagination.totalItems)
  : setTotalItems(employeesData.users.length);
} catch (error) {
  addNotification("Erro ao buscar funcionários", "error");
  setEmployees([]);
  setTotalItems(0);
}
};

const handleMakeAdmin = async () => {
  if (!selectedEmployee) return;

  setIsLoading(true);
  try {
    await adminService.changeUserRole(selectedEmployee.id, 0);
    addNotification(
      "Funcionário promovido a administrador com sucesso!",
      "success"
    );
    await fetchEmployees();

    handleClosePopup();
  } catch (error) {
    addNotification("Erro ao promover funcionário a administrador", "error");
  } finally {
    setIsLoading(false);
  }
};

const handleEmployeeClick = (employee: User) => {
  setSelectedEmployee(employee);
  setIsPopupOpen(true);
};

const handleClosePopup = () => {
  setIsPopupOpen(false);
```

```

    setSelectedEmployee(null);
};

const handleClosePopupCopy = () => {
  setIsPopupCopyOpen(false);
  setCode("null");
};

const handleAddEmployee = async () => {
  const code = await adminService.getEmployeeRegisterCode();
  setCode(
    `http://localhost:5173/registrar?token=${encodeURIComponent(code)}`);
};

setIsPopupCopyOpen(true);
};

const handlePageChange = (page: number) => {
  setCurrentPage(page);
  window.scrollTo({ top: 0, behavior: "smooth" });
};

useEffect(() => {
  const timer = setTimeout(() => {
    setDebouncedSearchTerm(searchTerm);
    setCurrentPage(1);
  }, 500);

  return () => clearTimeout(timer);
}, [searchTerm]);

useEffect(() => {
  fetchEmployees();
}, [currentPage, debouncedSearchTerm]);

const handleCopy = async () => {
  constisOk = await copyToClipboard(code);
  if (isOk) addNotification("Texto copiado com sucesso!", "success");
};

return (
<>
  <Sidebar />
  <Popup
    isOpen={isPopupCopyOpen}

```

```
onClose={handleClosePopupCopy}
title={"Adicionar novo funcionário"}
size="medium"
>
<ContactContainer>
  <p>Copie o link abaixo e mande para o seu funcionário</p>
  <CopyTextContainer>
    <TextToCopy>{code}</TextToCopy>
    <CopyButton onClick={handleCopy} />
  </CopyTextContainer>
</ContactContainer>
</Popup>
<Popup
  isOpen={isPopupOpen}
  onClose={handleClosePopup}
  title={selectedEmployee?.name || "Funcionário"}
  size="medium"
>
<ContactContainer>
  <p>Contato:</p>
  <DataContainer>
    <HiOutlineEnvelope /> {selectedEmployee?.email}
  </DataContainer>
  <DataContainer>ID: {selectedEmployee?.id}</DataContainer>
  <ButtonContainer>
    <CustomButton
      title={isLoading ? "Processando..." : "Tornar Administrador"}
      onPress={handleMakeAdmin}
      disabled={isLoading}
    />
  </ButtonContainer>
</ContactContainer>
</Popup>
<Container className="containerSide">
  <QueueHeader
    queueTitle="Funcionários"
    queueSubtitle="Lista de todos os funcionários"
    searchTerm={searchTerm}
    onChange={setSearchTerm}
    canSearch
    placeholder="Pesquisar funcionários"
  />
<EmployeContainer>
```

```

{employees.length > 0 ? (
  employees.map((employee) => (
    <Employee
      key={employee.id}
      employee={employee}
      onClick={() => handleEmployeeClick(employee)}
    />
  ))
) : (
  <NoEmployeesMessage>
    nenhum funcionário encontrado
  </NoEmployeesMessage>
)
)
</EmployeeContainer>

{totalItems > 0 && (
  <Pagination
    totalItems={totalItems}
    itemsPerPage={itemsPerPage}
    currentPage={currentPage}
    onPageChange={handlePageChange}
    maxVisibleButtons={5}
  />
)
}

<AddButton onClick={handleAddEmployee}>
  <HiPlus />
</AddButton>
</Container>
</>
);
}

```

styles.ts

```

import { color } from "@hisius/ui/theme/colors";
import { fadeInAnim } from "../../../../../assets/animations";
import styled from "styled-components";

export const Container = styled.div`
  display: flex;
  position: relative;
  min-height: 100%;
  flex-direction: column;

```

```
gap: 1rem;
';

export const ButtonContainer = styled.div`  
  display: flex;  
  margin-top: 2rem;  
  justify-content: center;  
  width: 100%;  
`;

export const ItemsPerPageSelector = styled.div`  
  display: flex;  
  align-items: center;  
  gap: 0.5rem;  
  margin-bottom: 1rem;  
  
  label {  
    font-size: 0.9rem;  
    color: ${color.gray};  
  }  
  
  select {  
    padding: 0.25rem 0.5rem;  
    border-radius: 4px;  
    border: 1px solid ${color.gray};  
    background-color: white;  
  }  
';

export const NoEmployeesMessage = styled.div`  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  height: 200px;  
  color: ${color.text};  
  font-size: 16px;  
  width: 100%;  
  text-align: center;  
  grid-column: 1 / -1;  
';

export const EmployeContainer = styled.div`  
  display: grid;
```

```
gap: 1rem;
place-items: center;
grid-template-columns: repeat(auto-fit, minmax(min(100%, 130px), 1fr));

@media (min-width: 1200px) {
  grid-template-columns: repeat(8, 1fr);
}

${fadeInAnim}
';

export const ContactContainer = styled.div`width: fit-content;
margin: 2rem auto;
`;

export const CopyTextContainer = styled.div`display: flex;
margin-top: 2rem;
justify-content: space-between;
background-color: ${color.background};
border: 1px solid ${color.gray};
border-radius: 4px;
padding: 0.5rem;
align-items: center;
`;

export const TextToCopy = styled.span`flex: 1;
width: 100px;
overflow: hidden;
text-overflow: ellipsis;
white-space: nowrap;
margin-right: 1rem;
color: ${color.text};
`;

export const DataContainer = styled.div`display: flex;
gap: 1rem;
margin-top: 1rem;

& svg {
  width: 20px;
  height: 20px;
}
```

```
}

`;

export const AddButton = styled.button`  
    width: 60px;  
    aspect-ratio: 1/1;  
    border-radius: 50%;  
    border: none;  
    cursor: pointer;  
  
    display: flex;  
    place-items: center;  
  
    position: fixed;  
  
    right: 5%;  
    bottom: 5%;  
  
    z-index: 9;  
  
    padding: 1rem;  
  
    &:active {  
        transform: scale(0.98);  
        transition: all 0.1s ease;  
    }  
  
    background-color: ${color.primary};  
    color: ${color.front};  
  
    & svg {  
        width: 100%;  
        height: 100%;  
        color: ${color.front};  
        aspect-ratio: 1/1;  
    }  
`;

export const TitleInfo = styled.div`  
    font-size: 16px;  
    font-weight: 200;  
`;

export const SubtitleInfo = styled.div`
```

```
font-size: 24px;
font-weight: 300;
';

export const TextContainer = styled.div`  

  display: flex;  

  flex-direction: column;  

  justify-content: center;  

  align-items: center;  

  gap: 1rem;
`;

export const InfoIcon = styled.div`  

  & svg {  

    width: 50px;  

    height: 50px;  

    color: ${color.text};  

  }
`;

export const QueueContainer = styled.div`  

  display: flex;  

  gap: 1rem;
`;

export const InfoCardContainer = styled.div`  

  display: flex;  

  width: 100%;  

  justify-content: space-between;
`;

export const SectionTitle = styled.h1`  

  margin-top: 2rem;  

  font-size: 24px;  

  font-weight: 400;
`;

export const InfoContainer = styled.div`  

  display: flex;  

  gap: 1rem;  

  justify-content: space-around;  

  align-items: center;  

  padding: 1.5rem;  

  background: ${color.front};
`;
```

```
border: 0.7px solid rgba(13, 19, 41, 0.12);
border-radius: 5px;
';
```

index.tsx

```
import type { IEmployee } from "@hisius/interfaces";
import { Container, Name, ViewMore } from "./styles";
interface EmployeeProps {
  employee: IEmployee;
  onClick: () => void;
}

export function Employee({ employee, onClick }: EmployeeProps) {
  return (
    <Container onClick={onClick}>
      <Name>{employee.name}</Name>
      <ViewMore>Ver Mais</ViewMore>
    </Container>
  );
}
```

styles.ts

```
import { color } from "@hisius/ui/theme/colors";
import { clickAnim, tabAnim } from "../../../../../assets/animations";
import styled, { css } from "styled-components";

export const Container = styled.div`
  width: 100%;
  max-width: 115px;
  background-color: ${color.front};
  border-radius: 5px;
  display: flex;
  flex-direction: column;
  justify-content: center;
  padding: 22px 20px;
  position: relative;
  gap: 2rem;
  transform-origin: top center;
  transition: all 0.3s ease-out;

  cursor: pointer;

  &:hover {
    transform: scale(1.005);
  }
`;
```

```

    box-shadow: 0px 4px 5px rgba(0, 0, 0, 0.05);
}
z-index: 1;

${clickAnim}
';

export const ViewMore = styled.button`
font-size: 13px;
font-weight: 200;
background: none;
border-radius: 5px;
border: 1px solid ${color.secondary};
cursor: pointer;
`;

export const Name = styled.h1`
transition: all ease 0.4s;
font-size: 13px;
font-weight: 400;
${css`
${Container}:hover & {
${tabAnim}
}
`}
`;

```

index.tsx

```

import { useEffect, useState } from "react";
import { QueueHeader } from "../../../../../components/navbar";
import { Sidebar } from "../../../../../components/sidebar";
import { Admin } from "@hisius/services/src";
import type { LogData } from "packages/interfaces/src";
import { useNotification } from "../../../../../components/notification/context";
import Pagination from "../../../../../components/pagination";
import {
  Container,
  HeaderRow,
  SheetValue,
  Row,
  RowValues,
  LoadingText,
  NoDataText,
} from "./styles";
import { usePageTitle } from "../../../../../hooks/PageTitle";

export default function LogsList() {

```

```

const [searchTerm, setSearchTerm] = useState("");
const [debouncedSearchTerm, setDebouncedSearchTerm] = useState("");
const [logs, setLogs] = useState<LogData[]>([]);
const [currentPage, setCurrentPage] = useState(1);
const [totalItems, setTotalItems] = useState(0);
const [itemsPerPage] = useState(12);
const [loading, setLoading] = useState<boolean>(true);

const adminService = new Admin();
const { addNotification } = useNotification();

usePageTitle("Atividades - Hisius");

useEffect(() => {
  const timer = setTimeout(() => {
    setDebouncedSearchTerm(searchTerm);
    setCurrentPage(1);
  }, 500);

  return () => clearTimeout(timer);
}, [searchTerm]);

useEffect(() => {
  const fetchLogs = async () => {
    try {
      setLoading(true);
      const apiPage = currentPage - 1;

      const searchAction =
        debouncedSearchTerm.length >= 3 ? debouncedSearchTerm : undefined;

      const logsResponse = await adminService.getLogs(
        apiPage,
        itemsPerPage,
        undefined,
        searchAction
      );

      setLogs(logsResponse.logs);

      logsResponse.pagination
        ? setTotalItems(logsResponse.pagination.totalItems)
        : setTotalItems(logsResponse.logs.length);
    } catch (error) {
  
```

```
    addNotification("Erro ao buscar logs", "error");
} finally {
  setLoading(false);
}
};

fetchLogs();
}, [currentPage, debouncedSearchTerm]);

const handlePageChange = (page: number) => {
  setCurrentPage(page);
  window.scrollTo({ top: 0, behavior: "smooth" });
};

const formatDate = (dateString: string) => {
  return new Date(dateString).toLocaleString("pt-BR");
};

return (
<>
<Sidebar />

<div className="containerSide">
<QueueHeader
  queueTitle="Atividades recentes"
  queueSubtitle="Veja as atividades recentes"
  searchTerm={searchTerm}
  onChange={setSearchTerm}
  canSearch
  placeholder="Pesquisar atividades..."
/>

<Container>
<HeaderRow>
  <SheetValue>Ação</SheetValue>
  <SheetValue>ID Usuário</SheetValue>
  <SheetValue>Módulo</SheetValue>
  <SheetValue>Data</SheetValue>
</HeaderRow>

{loading ? (
  <LoadingText>Carregando logs...</LoadingText>
) : logs.length === 0 ? (
  <NoDataText>
```

```

{debouncedSearchTerm.length >= 3
 ? "Nenhum log encontrado para a pesquisa"
 : "Nenhum log disponível"
</NoDataText>
):(
<>
{logs.map((log) => (
<Row key={log.id}>
<RowValues>{log.action}</RowValues>
<RowValues>{log.userId}</RowValues>
<RowValues>{log.module}</RowValues>
<RowValues>{formatDate(log.createdAt)}</RowValues>
</Row>
))}

<Pagination
totalItems={totalItems}
itemsPerPage={itemsPerPage}
currentPage={currentPage}
onPageChange={handlePageChange}
maxVisibleButtons={5}
/>
</>
)}
</Container>
</div>
</>
);
}

```

styles.ts

```

import styled from "styled-components";

export const HeaderRow = styled.div`
display: grid;
grid-template-columns: 2fr 1fr 1fr 1fr;
margin-top: 30px;
gap: 1rem;
`;

export const SheetValue = styled.div`
background-color: white;
box-shadow: 0px 2px 2px #00000010;

```

```

padding: 20px;
border-radius: 5px;
';

export const Row = styled.div`
display: grid;
grid-template-columns: 2fr 1fr 1fr 1fr;
gap: 1rem;
margin-top: 8px;
padding: 16px;
border-bottom: 0.7px solid #0000003b;
`;

export const RowValues = styled.div`
font-weight: 300;
padding: 0 15px;
font-size: 14px;
`;

export const Container = styled.div`
padding: 20px;
`;

export const LoadingText = styled.div`
text-align: center;
padding: 20px;
font-style: italic;
color: #666;
`;

export const NoDataText = styled.div`
text-align: center;
padding: 20px;
color: #666;
`;

```

index.tsx

```

import { color } from "@hisius/ui/theme/colors";
import { QueueHeader } from "../../../../../components/navbar";
import { Sidebar } from "../../../../../components/sidebar";
import SimpleBarChart from "./SimplesBarChart";
import {
  Container,

```

```

GraphContainer,
GraphHourContainer,
GraphTitle,
HourContainer,
PeakContainer,
EmptyStateContainer,
EmptyStateText,
} from "./styles";
import DonutChart from "./DonutChart";
import { useEffect, useState } from "react";
import { Admin } from "@hisius/services";
import { useNotification } from "../../../../../components/notification/context";
import type { ReportInfo } from "@hisius/interfaces";
import { formatTime } from "../../../../../utils";
import Toggle from "../../../../../employee/components/toggle";
import { usePageTitle } from "../../../../../hooks/PageTitle";

export function Report() {
  const colors = (): string[] => {
    const days = [
      "domingo",
      "segunda",
      "terça",
      "quarta",
      "quinta",
      "sexta",
      "sábado",
    ];
    const today = new Date().getDay();

    return days.map((_, index) =>
      index === today ? color.primary : color.text
    );
  };

  const formatDate = (date: Date) => {
    const day = String(date.getDate()).padStart(2, "0");
    const month = String(date.getMonth() + 1).padStart(2, "0");
    const year = date.getFullYear();
    return `${day}-${month}-${year}`;
  };

  const adminService = new Admin();
  const { addNotification } = useNotification();

```

```

const [isMonthly, setIsMonthly] = useState(false);
const [reportData, setReportData] = useState<ReportInfo>();
const [hasData, setHasData] = useState(false);

usePageTitle("Relatórios - Hisius");

const weekDays = ["DOM", "SEG", "TER", "QUA", "QUI", "SEX", "SAB"] as const;

const mappedPeakDemand = weekDays.map(
  (day) => reportData?.peakDemand[day] ?? 0
);

const checkDataAvailability = (data: ReportInfo | undefined): boolean => {
  if (!data) return false;
}

const hasAvgTimeTreatment =
  data.avgTimeTreatmentInSec &&
  data.avgTimeTreatmentInSec.length > 0 &&
  data.avgTimeTreatmentInSec.some((item) => item.count > 0);

const hasAvgTimeTriage =
  data.avgTimeTriageInSec &&
  data.avgTimeTriageInSec.length > 0 &&
  data.avgTimeTriageInSec.some((item) => item.count > 0);

const hasPeakDemand =
  data.peakDemand &&
  Object.values(data.peakDemand).some((value) => value > 0);

return hasAvgTimeTreatment || hasAvgTimeTriage || hasPeakDemand;
};

useEffect(() => {
  const fetchReportData = async () => {
    try {
      const today = new Date();
      let startDate: string;
      let endDate: string;

      if (isMonthly) {
        const startOfMonth = new Date(
          today.getFullYear(),
          today.getMonth(),
          1
        );
        const endOfMonth = new Date(
          startOfMonth.getFullYear(),
          startOfMonth.getMonth() + 1,
          1
        );
        startDate = startOfMonth.toISOString().split("T")[0];
        endDate = endOfMonth.toISOString().split("T")[0];
      } else {
        const startOfDay = new Date(
          today.getFullYear(),
          today.getMonth(),
          today.getDate(),
          0,
          0,
          0
        );
        const endOfDay = new Date(
          startOfDay.getFullYear(),
          startOfDay.getMonth(),
          startOfDay.getDate(),
          23,
          59,
          59
        );
        startDate = startOfDay.toISOString().split("T")[0];
        endDate = endOfDay.toISOString().split("T")[0];
      }
      const response = await fetch(`https://api.hisius.com.br/reports?start=${startDate}&end=${endDate}`);
      const data = await response.json();
      setReportData(data);
    } catch (error) {
      console.error(error);
    }
  };
}, []);

```

```

);

const endOfMonth = new Date(
  today.getFullYear(),
  today.getMonth() + 1,
  0
);
startDate = formatDate(startOfMonth);
endDate = formatDate(endOfMonth);
} else {
  const dayOfWeek = today.getDay();
  const startOfWeek = new Date(today);
  startOfWeek.setDate(today.getDate() - dayOfWeek);
  const endOfWeek = new Date(startOfWeek);
  endOfWeek.setDate(startOfWeek.getDate() + 6);
  startDate = formatDate(startOfWeek);
  endDate = formatDate(endOfWeek);
}

const hospitalInfo = await adminService.getReport(startDate, endDate);
setReportData(hospitalInfo);
setHasData(checkDataAvailability(hospitalInfo));
} catch (error) {
  addNotification("Erro ao buscar informações", "error");
  setHasData(false);
}
};

fetchReportData();
}, [isMonthly]);

const handleToggleChange = async (checked: boolean) => {
  setIsMonthly(checked);
};

const EmptyState = () => (
<EmptyStateContainer>
  <EmptyStateText>Dados insuficientes</EmptyStateText>
</EmptyStateContainer>
);

return (
<>
  <Sidebar />
  <Container className="containerSide">

```

```

<QueueHeader queueTitle="Relatórios" queueSubtitle="Semana | Mês" />

<Toggle
  labels={{ on: "Mensal", off: "Semanal" }}
  onToggle={handleToggleChange}
/>

{!hasData ? (
  <EmptyState />
) : (
  <>
    <HourContainer>
      <GraphHourContainer>
        <GraphTitle>Tempo médio de espera (atendimento):</GraphTitle>
        <GraphContainer>
          {reportData?.avgTimeTreatmentInSec &&
            reportData.avgTimeTreatmentInSec.length > 0 &&
            reportData.avgTimeTreatmentInSec.some(
              (item) => item.count > 0
            ) ? (
              <DonutChart
                labels={
                  reportData.avgTimeTreatmentInSec.map((item) =>
                    formatTime(item.averageWaitTime)
                  ) || []
                }
                data={
                  reportData.avgTimeTreatmentInSec.map(
                    (item) => item.count
                  ) || []
                }
                color={color.primary}
                height="100%"
              />
            ) : (
              <EmptyState />
            )}}
        </GraphContainer>
      </GraphHourContainer>
    </HourContainer>
  </>
)<GraphHourContainer>
  <GraphTitle>Tempo médio de espera (triagem):</GraphTitle>
  <GraphContainer>
    {reportData?.avgTimeTriageInSec &&

```

```

reportData.avgTimeTriageInSec.length > 0 &&
reportData.avgTimeTriageInSec.some(
  (item) => item.count > 0
) ? (
  <DonutChart
    labels={
      reportData.avgTimeTriageInSec.map((item) =>
        formatTime(item.averageWaitTime)
      ) || []
    }
    data={
      reportData.avgTimeTriageInSec.map(
        (item) => item.count
      ) || []
    }
    color={color.primary}
    width="300px"
    height="100%"
  />
) : (
  <EmptyState />
)
</GraphContainer>
</GraphHourContainer>
</HourContainer>

<PeakContainer>
  <GraphTitle>Pico de demanda:</GraphTitle>
  {reportData?.peakDemand &&
  Object.values(reportData.peakDemand).some(
    (value) => value > 0
  ) ? (
    <SimpleBarChart
      labels={weekDays.slice()}
      data={mappedPeakDemand}
      colors={colors()}
      height={300}
    />
  ) : (
    <EmptyState />
  )
  </PeakContainer>
</>
)

```

```
</Container>
</>
);
}
```

styles.ts

```
import { color } from "@hisius/ui/theme/colors";
import { subtleSlideAnim } from "../../../../../assets/animations";
import styled from "styled-components";

export const Container = styled.div`  
  display: flex;  
  position: relative;  
  min-height: 100%;  
  flex-direction: column;  
  gap: 1rem;  
`;

export const EmptyStateContainer = styled.div`  
  display: flex;  
  justify-content: center;  
  align-items: center;  
  height: 200px;  
  width: 100%;  
`;

export const EmptyStateText = styled.span`  
  font-size: 16px;  
  color: ${color.text};  
  font-weight: 500;  
`;

export const HourContainer = styled.div`  
  width: 100%;  
  min-height: 45%;  
  display: flex;  
  justify-content: space-between;  
  ${subtleSlideAnim}  
;

@media (max-width: 968px) {  
  flex-direction: column;  
  gap: 1rem;
```

```

        }

';

export const PeakContainer = styled.div` 
  width: 100%; 
  height: 30%; 
  display: flex; 
  flex-direction: column; 
  gap: 1rem; 
  padding: 1rem; 
  border-radius: 5px; 
  background-color: ${color.front}; 
  ${subtleSlideAnim}
`;

export const GraphContainer = styled.div` 
  width: 100%; 
  min-height: 300px; 
  place-items: center;
`;

export const GraphHourContainer = styled.div` 
  width: 48%; 
  background-color: ${color.front}; 
  border-radius: 5px; 
  display: flex; 
  flex-direction: column; 
  padding: 1rem; 
  gap: 1rem;
  
  @media (max-width: 968px) { 
    width: 100%; 
  }
`;

export const GraphTitle = styled.h2` 
  font-size: 14px; 
  font-weight: 200;
`;

```

index.tsx

```

import React from "react";
import { Chart as ChartJS, ArcElement, Tooltip, Legend } from "chart.js";

```

```
import { Doughnut } from "react-chartjs-2";
import type { ChartOptions } from "chart.js";
import { color } from "@hisius/ui/theme/colors";
import ChartDataLabels from "chartjs-plugin-datalabels";

ChartJS.register(ArcElement, Tooltip, Legend, ChartDataLabels);

interface DonutChartProps {
  labels: string[];
  data: number[];
  color?: string;
  height?: number | string;
  width?: number | string;
}

const DonutChart: React.FC<DonutChartProps> = ({  
  labels,  
  data,  
  color: mcolor = "rgba(54, 162, 235, 0.8)",  
  height = 300,  
  width = 300,  
) => {  
  const options: ChartOptions<"doughnut"> = {  
    responsive: true,  
    maintainAspectRatio: false,  
    font: {  
      family: "montserrat",  
      size: 13,  
      weight: 300,  
    },  
    cutout: "70%",  
    layout: {  
      padding: 40  
    },  
    plugins: {  
      legend: {  
        display: false,  
      },  
      tooltip: {  
        enabled: true,  
        backgroundColor: color.front,  
        titleColor: color.text,  
        bodyColor: color.text,  
        borderColor: color.gray,  
      },  
    },  
  };  
  return (  
    <Doughnut data={data} options={options} />  
  );  
};
```

```
borderWidth: 0.7,  
cornerRadius: 4,  
displayColors: false,  
callbacks: {  
    title: () => {  
        return ``;  
    },  
    label: (context) => {  
        return `Valor: ${context.parsed}`;  
    },  
},  
},  
datalabels: {  
    color: color.text,  
    formatter: (_value, context) =>  
        context.chart.data.labels?.[context.dataIndex],  
    anchor: "end",  
    align: "end",  
    display: true,  
    font: {  
        family: "montserrat",  
        size: 13,  
        weight: 300,  
    },  
},  
},  
elements: {  
    arc: {  
        borderWidth: 2,  
        borderColor: "#fff",  
    },  
},  
};  
  
const chartData = {  
    labels,  
    datasets: [  
        {  
            data,  
            backgroundColor: Array(data.length).fill(mcolor),  
            borderWidth: 2,  
            borderColor: "#fff",  
        },  
    ],
```

```

};

return (
<div
  style={{
    height: `${height}`,
    width: `${width}`,
    position: "relative",
  }}
>
  <Doughnut data={chartData} options={options} />
</div>
);
};

export default DonutChart;

```

index.tsx

```

import React from "react";
import {
  Chart as ChartJS,
  CategoryScale,
  LinearScale,
  BarElement,
  Tooltip,
} from "chart.js";
import ChartDataLabels from "chartjs-plugin-datalabels";
import { Bar } from "react-chartjs-2";
import type { ChartOptions } from "chart.js";
import { color } from "@hisius/ui/theme/colors";

ChartJS.register(
  CategoryScale,
  LinearScale,
  BarElement,
  Tooltip,
  ChartDataLabels
);

interface ShortBarChartProps {
  labels: string[];
  data: number[];
  colors?: string[];
}

```

```
height?: number | string;
}

const ShortBarChart: React.FC<ShortBarChartProps> = ({  
  labels,  
  data,  
  colors = [  
    "rgba(54, 162, 235, 0.8)",  
    "rgba(54, 162, 235, 0.8)",  
    "rgba(54, 162, 235, 0.8)",  
    "rgba(255, 99, 132, 0.8)",  
    "rgba(54, 162, 235, 0.8)",  
    "rgba(54, 162, 235, 0.8)",  
    "rgba(54, 162, 235, 0.8)",  
    "rgba(54, 162, 235, 0.8)",  
  ],  
  height = 150,  
) => {  
  const options: ChartOptions<"bar"> = {  
    responsive: true,  
    maintainAspectRatio: false,  
    animation: {  
      duration: 1000,  
      easing: "easeOutQuart",  
    },  
    font: {  
      family: "montserrat",  
      size: 13,  
      weight: 300,  
    },  
    plugins: {  
      legend: {  
        display: false,  
      },  
      tooltip: {  
        enabled: true,  
        backgroundColor: color.background,  
        titleColor: color.text,  
        bodyColor: color.text,  
        borderColor: color.gray,  
        borderWidth: 0.7,  
        cornerRadius: 4,  
        displayColors: false,  
        callbacks: {  
          title: (context) => {  
            return context.dataset.data[context.dataIndex];  
          },  
        },  
      },  
    },  
  };  
  return (  
    <BarChart data={data} labels={labels} colors={colors} options={options} />  
  );  
}
```

```
        return `Dia: ${context[0].label}`;
    },
    label: (context) => {
        return `Valor: ${context.parsed.y}`;
    },
},
},
datalabels: {
    display: false,
},
},
scales: {
    x: {
        beginAtZero: true,
        grid: {
            display: false,
        },
        ticks: {
            color: "#666",
        },
        border: {
            display: true,
            color: `${color.text}`,
            width: 1,
        },
    },
    y: {
        beginAtZero: true,
        grid: {
            display: false,
        },
        border: {
            display: true,
            color: `${color.text}`,
            width: 1,
        },
        ticks: {
            display: true,
            font: {
                size: 13,
            },
        },
    },
},
},
```

```

elements: {
  bar: {
    borderRadius: 5,
    borderSkipped: false,
  },
},
};

const chartData = {
  labels,
  datasets: [
    {
      data,
      backgroundColor: colors,
      borderWidth: 0,
      barPercentage: 0.5,
      categoryPercentage: 0.7,
    },
  ],
};

return (
  <div style={{ height: `${height}px`, width: "100%" }}>
    <Bar data={chartData} options={options} />
  </div>
);
};

export default ShortBarChart;

```

index.ts

```

export const capitalizeWords = (value: string | undefined) => {
  if (!value) return "";

  return value
    .split(" ")
    .map((word) => word.charAt(0).toUpperCase() + word.slice(1))
    .join(" ");
};

export async function copyToClipboard(text: string): Promise<boolean> {
  try {
    await navigator.clipboard.writeText(text);
    return true;
  }
}

```

```

    } catch (err) {
      return false;
    }
  }
export const formatTime = (seconds: number): string => {
  const hours = Math.floor(seconds / 3600);
  const minutes = Math.floor((seconds % 3600) / 60);
  const secs = seconds % 60;

  return hours
    ? `${hours}h${minutes ? ` ${minutes}min` : ""}`
    : minutes
      ? `${minutes}min`
      : `${secs}s`;
};

```

MOBILE

MOBILEindex.tsx

```

import React, { forwardRef } from "react";
import { TextInput, TextInputProps } from "react-native";

export const CodeInputBase = forwardRef<TextInput, TextInputProps>(
  (props, ref) => <TextInput ref={ref} {...props} />
);

```

index.tsx

```

import React, { useState } from "react";
import {
  View,
  Text,
  TouchableOpacity,
  Modal,
  FlatList,
  StyleProp,
  TextStyle,
} from "react-native";
import { styles } from "./styles";
import Icon from "react-native-vector-icons/Feather";
import { color } from "packages/ui/theme/colors";

interface CustomPickerProps {
  value: string;

```

```

onValueChange: (value: string) => void;
options: { label: string; value: string }[];
placeholder?: string;
error?: string;
icon?: React.ReactNode;
style?: StyleProp<TextStyle>;
}

const CustomPicker: React.FC<CustomPickerProps> = ({
  value,
  onValueChange,
  options,
  placeholder = "Seleccione",
  error = "",
  icon,
  style,
}) => {
  const [modalVisible, setModalVisible] = useState(false);
  const selectedLabel =
    options.find((opt) => opt.value === value)?.label || placeholder;

  return (
    <View>
      <TouchableOpacity
        style={[
          styles.inputContainer,
          error ? styles.inputContainerError : null,
        ]}
        onPress={() => setModalVisible(true)}
        activeOpacity={0.7}
      >
        {icon && <View style={styles.iconContainer}>{icon}</View>}
        <Text style={[styles.textValue, style]}>{selectedLabel}</Text>
        <View style={styles.dropdownIcon}>
          <Icon name="chevron-down" size={20} color={color.text} />
        </View>
      </TouchableOpacity>
    <Text style={styles.errorText}>{error}</Text> : null
  )
}

<Modal
  visible={modalVisible}
  transparent
  animationType="fade"
/>

```

```

onRequestClose={() => setModalVisible(false)}
>
<TouchableOpacity
  style={styles.modalOverlay}
  activeOpacity={1}
  onPressOut={() => setModalVisible(false)}
>
  <View style={styles.modalContent}>
    <FlatList
      data={options}
      keyExtractor={(item) => item.value}
      renderItem={({ item }) => (
        <TouchableOpacity
          onPress={() => {
            onValueChange(item.value);
            setModalVisible(false);
          }}
          style={styles.item}
        >
          <Text style={styles.itemText}>{item.label}</Text>
        </TouchableOpacity>
      )}
    />
  </View>
</TouchableOpacity>
</Modal>
</View>
);
};

export default CustomPicker;

```

styles.tsx

```

import { StyleSheet } from "react-native";

export const styles = StyleSheet.create({
  inputContainer: {
    flexDirection: "row",
    alignItems: "center",
    borderWidth: 1,
    borderColor: "#ddd",
    borderRadius: 8,
    backgroundColor: "#fff",
  }
});

```

```
height: 50,  
paddingHorizontal: 12,  
shadowColor: "#000",  
shadowOffset: { width: 0, height: 1 },  
shadowOpacity: 0.05,  
shadowRadius: 3,  
},  
inputContainerError: {  
  borderColor: "#FF3B30",  
},  
textValue: {  
  flex: 1,  
  fontSize: 16,  
  color: "#333",  
},  
iconContainer: {  
  marginRight: 8,  
  justifyContent: "center",  
  alignItems: "center",  
},  
dropdownIcon: {  
  marginLeft: 8,  
  justifyContent: "center",  
  alignItems: "center",  
},  
errorText: {  
  color: "#FF3B30",  
  fontSize: 14,  
  marginTop: 4,  
  marginLeft: 4,  
},  
modalOverlay: {  
  flex: 1,  
  backgroundColor: "rgba(0,0,0,0.2)",  
  justifyContent: "center",  
},  
modalContent: {  
  marginHorizontal: 20,  
  backgroundColor: "#fff",  
  borderRadius: 8,  
  maxHeight: "50%",  
  paddingVertical: 8,  
  shadowColor: "#000",  
  shadowOffset: { width: 0, height: 1 },
```

```

    shadowOpacity: 0.1,
    shadowRadius: 3,
  },
  item: {
    paddingVertical: 12,
    paddingHorizontal: 16,
  },
  itemText: {
    fontSize: 16,
    color: "#333",
  },
);

```

index.tsx

```

import { color } from "@hisius/ui/theme/colors";
import { Text, TextProps } from "react-native";

export function GlobalText(props: TextProps) {
  return (
    <Text
      {...props}
      style={[{ fontFamily: "Montserrat", color: color.text }, props.style]}
    />
  );
}

```

index.tsx

```

import React from "react";
import { View } from "react-native";
import Icon from "react-native-vector-icons/Feather";
import { ButtonText, Container, ProfileButton, SoftwareName } from "./styles";

interface HeaderProps {
  softwareName: string;
  onProfilePress?: () => void;
}

const Header: React.FC<HeaderProps> = ({ softwareName, onProfilePress }) => {
  return (
    <Container>
      <SoftwareName>{softwareName}</SoftwareName>
    </Container>
  );
}

```

```

<ProfileButton onPress={onProfilePress}>
  <View style={{ flexDirection: "row", alignItems: "center" }}>
    <Icon name="user" size={14} color="#000" style={{ marginRight: 6 }} />
    <ButtonText>Perfil</ButtonText>
  </View>
</ProfileButton>
</Container>
);
};

export default Header;

```

styles.tsx

```

import styled from "styled-components/native";
import { TouchableOpacity, Text } from "react-native";
import { GlobalText } from "../globalText";
import { color } from "@hisius/ui/theme/colors";
import { scale } from "../../utils/scale";

export const Container = styled.View`
  flex-direction: row;
  width: 100%;
  justify-content: space-between;
  align-items: center;
  padding: 12px 20px;
  background-color: transparent;
`;

export const SoftwareName = styled(GlobalText)`
  font-size: ${scale(25)};
  font-weight: bold;
  text-transform: uppercase;
`;

export const ProfileButton = styled(TouchableOpacity)`
  padding: 6px 12px;

  border-color: ${color.gray};
  background-color: ${color.card};

  border-radius: 15px;
  border-width: 1px;
`;

```

```
export const ButtonText = styled(GlobalText)`  
  font-size: 16px;  
  color: ${color.text};  
`;
```

container.tsx

```
import React from "react";  
import { useNotification } from "./context";  
import { NotificationContainerWrapper } from "./styles";  
import { NotificationComponent } from ".";  
  
export const NotificationContainer: React.FC = () => {  
  const { notifications, removeNotification } = useNotification();  
  
  if (notifications.length === 0) {  
    return null;  
  }  
  
  return (  
    <NotificationContainerWrapper>  
      {notifications.map((notification) => (  
        <NotificationComponent  
          key={notification.id}  
          notification={notification}  
          onClose={() => removeNotification(notification.id)}  
        />  
      ))}  
    </NotificationContainerWrapper>  
  );  
};
```

context.tsx

```
import React, { createContext, useContext, useState, useCallback } from "react";  
import type {  
  NotificationContextType,  
  NotificationProviderProps,  
  Notification,  
} from "./types";  
import { NotificationContainer } from "./container";
```

```

const NotificationContext = createContext<NotificationContextType | undefined>(
  undefined
);

export const useNotification = (): NotificationContextType => {
  const context = useContext(NotificationContext);
  if (!context) {
    throw new Error(
      "useNotification deve ser usado dentro de um NotificationProvider"
    );
  }
  return context;
};

export const NotificationProvider: React.FC<NotificationProviderProps> = ({
  children,
}) => {
  const [notifications, setNotifications] = useState<Notification[]>([]);

  const addNotification = useCallback(
    (
      message: string,
      type: Notification["type"] = "info",
      duration: number = 5000
    ): string => {
      const id = Date.now() + Math.random().toString();
      const newNotification: Notification = {
        id,
        message,
        type,
        duration,
      };
      setNotifications((prev) => [...prev, newNotification]);

      if (duration > 0) {
        setTimeout(() => {
          removeNotification(id);
        }, duration);
      }
    },
    []
  );

  return id;
};

```

```

);

const removeNotification = useCallback((id: string) => {
  setNotifications((prev) =>
    prev.filter((notification) => notification.id !== id)
  );
}, []);

const clearAll = useCallback(() => {
  setNotifications([]);
}, []);

const value: NotificationContextType = {
  notifications,
  addNotification,
  removeNotification,
  clearAll,
};

return (
  <NotificationContext.Provider value={value}>
    {children}
    <NotificationContainer />
  </NotificationContext.Provider>
);
};

```

index.tsx

```

import React, { useEffect, useState, useRef } from "react";
import { Animated, Text, ViewStyle } from "react-native";
import type { NotificationComponentProps } from "./types";
import {
  NotificationWrapper,
  NotificationContent,
  NotificationMessage,
  CloseButton,
  CloseButtonContent,
} from "./styles";

export const NotificationComponent: React.FC<NotificationComponentProps> = ({ notification, onClose, style, });

```

```

}) => {
  const { message, type, duration } = notification;
  const [isExiting, setIsExiting] = useState<boolean>(false);
  const slideAnim = useRef(new Animated.Value(0)).current;
  const fadeAnim = useRef(new Animated.Value(0)).current;

  useEffect(() => {
    Animated.parallel([
      Animated.timing(slideAnim, {
        toValue: 1,
        duration: 300,
        useNativeDriver: true,
      }),
      Animated.timing(fadeAnim, {
        toValue: 1,
        duration: 300,
        useNativeDriver: true,
      }),
    ]).start();
  }, []);

  const handleClose = () => {
    setIsExiting(true);
    Animated.parallel([
      Animated.timing(slideAnim, {
        toValue: 0,
        duration: 250,
        useNativeDriver: true,
      }),
      Animated.timing(fadeAnim, {
        toValue: 0,
        duration: 250,
        useNativeDriver: true,
      }),
    ]).start(() => {
      onClose();
    });
  };

  useEffect(() => {
    if (duration > 0) {
      const timer = setTimeout(() => {
        handleClose();
      }, duration - 300);
    }
  });
}

```

```

    return () => clearTimeout(timer);
}
}, [duration]);
}

const slideInterpolate = slideAnim.interpolate({
  inputRange: [0, 1],
  outputRange: [100, 0],
});

const animatedStyle = {
  transform: [{ translateX: slideInterpolate }],
  opacity: fadeAnim,
};

const nativeStyle = style as ViewStyle;

return (
  <Animated.View style={[animatedStyle, nativeStyle]}>
    <NotificationWrapper $type={type} $isExiting={isExiting}>
      <NotificationContent>
        <NotificationMessage>{message}</NotificationMessage>
        <CloseButton onPress={handleClose}>
          <CloseButtonContent>
            <Text style={{ fontSize: 18, color: "#000" }}>x</Text>
          </CloseButtonContent>
        </CloseButton>
      </NotificationContent>
    </NotificationWrapper>
  </Animated.View>
);
};

```

styles.ts

```

import styled, { css } from "styled-components/native";
import { Dimensions } from "react-native";
import type { NotificationWrapperProps } from "./types";
import { color } from "@hisius/ui/theme/colors";

export const NotificationContainerWrapper = styled.View` 
  position: absolute;
  bottom: 20px;
  right: 20px;
  z-index: 1000;

```

```
display: flex;
flex-direction: column-reverse;
gap: 16px;

${() => {
  const { width } = Dimensions.get("window");
  if (width <= 768) {
    return css`
      right: 10px;
      left: 10px;
    `;
  }
  return "";
}}

export const NotificationWrapper = styled.View<NotificationWrapperProps>`  

  min-width: 300px;  

  max-width: 400px;  

  padding: 16px;  

  position: relative;  

  border-left-width: 6px;  

  border-left-color: ${color.front};  

  background-color: ${color.front};  

  border-radius: 5px;  

  shadow-color: #000;  

  shadow-offset: 0px 4px;  

  shadow-opacity: 0.15;  

  shadow-radius: 12px;  

  elevation: 5;

${(props) => {
  switch (props.$type) {
    case "info":
      return css`
        border-left-color: ${color.primary};
      `;
    case "error":
      return css`
        border-left-color: ${color.error.error};
      `;
    case "warning":
      return css`
        border-left-color: ${color.error.warning};
      `;
```

```
`;  
case "success":  
    return css`  
        border-left-color: ${color.error.ok};  
`;  
default:  
    return css`  
        border-left-color: ${color.primary};  
`;  
}  
};  
  
${() => {  
    const { width } = Dimensions.get("window");  
    if (width <= 768) {  
        return css`  
            min-width: auto;  
            max-width: none;  
`;  
    }  
    return "";  
}}  
;  
  
export const NotificationContent = styled.View`  
    display: flex;  
    flex-direction: row;  
    justify-content: space-between;  
    align-items: flex-start;  
    gap: 16px;  
`;  
  
export const NotificationMessage = styled.Text`  
    font-size: 14px;  
    flex: 1;  
    margin: 0;  
    color: #000;  
    font-weight: 300;  
`;  
  
export const CloseButton = styled.TouchableOpacity`  
    background-color: transparent;  
    border: none;  
    padding: 0;  
`;
```

```
width: 24px;
height: 24px;
display: flex;
align-items: center;
justify-content: center;
border-radius: 12px;
opacity: 0.7;
';

export const CloseButtonContent = styled.View`  
width: 100%;  
height: 100%;  
align-items: center;  
justify-content: center;  
border-radius: 12px;  
`;
```

types.ts

```
import type { ReactNode } from "react";

export type NotificationType = "info" | "error" | "warning" | "success";

export interface Notification {
  id: string;
  message: string;
  type: NotificationType;
  duration: number;
}

export interface NotificationContextType {
  notifications: Notification[];
  addNotification: (
    message: string,
    type?: NotificationType,
    duration?: number
  ) => string;
  removeNotification: (id: string) => void;
  clearAll: () => void;
}

export interface NotificationProviderProps {
  children: ReactNode;
}
```

```
export interface NotificationComponentProps {
  notification: Notification;
  onClose: () => void;
  style?: React.CSSProperties;
}
```

```
export interface NotificationWrapperProps {
  $type: NotificationType;
  $isExiting?: boolean;
}
```

index.tsx

```
import type { ApiError, ApiResponse } from "@hisius/interfaces/src";
import { useState, useCallback } from "react";
```

```
export const useFormErrors = () => {
  const [errors, setErrors] = useState<Record<string, string>>({});
```

```
const setFieldError = useCallback((field: string, message: string) => {
  setErrors((prev) => ({ ...prev, [field]: message }));
}, []);
```

```
const clearFieldError = useCallback((field: string) => {
  setErrors((prev) => {
    const newErrors = { ...prev };
    delete newErrors[field];
    return newErrors;
  });
}, []);
```

```
const clearAllErrors = useCallback(() => {
  setErrors({});
}, []);
```

```
const handleApiErrors = useCallback((errorResponse: ApiResponse) => {
  const newErrors: Record<string, string> = {};

  if (errorResponse.errors?.length) {
    errorResponse.errors.forEach((error: ApiError) => {
      newErrors[error.field] = error.message;
    });
  }
}
```

```

    setErrors(newErrors);
}, []);
```

```

return {
  errors,
  setFieldError,
  clearFieldError,
  clearAllErrors,
  handleApiErrors,
};
```

```

};
```

style.tsx

```

import styled from "styled-components/native";
import { Dimensions, TouchableOpacityProps } from "react-native";
import { GlobalText } from "../components/globalText";
import { color } from "@hisius/ui/theme/colors";

const { width, height } = Dimensions.get("window");

interface ModalContentProps {
  height?: number;
}

interface ButtonProps extends TouchableOpacityProps {
  primary?: boolean;
}

interface InputProps {
  hasError?: boolean;
}

interfaceInputLabelProps {
  type?: "hint" | "error" | "default";
}

export const ModalOverlay = styled.View`  

  flex: 1;  

  background-color: rgba(0, 0, 0, 0.5);  

  justify-content: center;  

  align-items: center;  

  padding: 20px;  

`;
```

```

export const ModalContent = styled.View<ModalContentProps>`  

  background-color: white;
```

```
border-radius: 12px;  
width: ${width - 40}px;  
max-height: ${(props) => props.height || height * 0.8}px;  
';  
  
export const ModalTitle = styled(GlobalText)`  
font-size: 18px;  
font-weight: bold;  
flex: 1;  
';  
  
export const CloseButton = styled.TouchableOpacity`  
padding: 5px;  
margin-left: 10px;  
';  
  
export const CloseButtonText = styled(GlobalText)`  
font-size: 20px;  
color: ${color.gray};  
font-weight: bold;  
';  
  
export const ModalBody = styled.ScrollView`  
padding: 20px;  
';  
  
export const ModalFooter = styled.View`  
padding: 15px 20px;  
border-top-width: 1px;  
border-top-color: ${color.front};  
flex-direction: row;  
justify-content: flex-end;  
gap: 10px;  
';  
  
export const Button = styled.TouchableOpacity<ButtonProps>`  
padding: 12px 24px;  
border-radius: 6px;  
background-color: ${(props) => (props.primary ? color.primary : color.front)};  
min-width: 80px;  
align-items: center;  
';  
  
export const ButtonText = styled.Text<{ primary?: boolean }>`  
color: ${(props) => (props.primary ? color.front : color.text)};  
font-size: 16px;  
font-weight: 500;  
';
```

```
export const InputContainer = styled.View`  
  margin: 35px 0;  
`;  
  
export const Input = styled.TextInput<InputProps>`  
  border-width: 1px;  
  border-color: ${(props) =>  
    props.hasError ? color.error.error : color.background};  
  border-radius: 8px;  
  padding: 12px 15px;  
  font-size: 16px;  
  background-color: ${color.background};  
  font-family: "Montserrat";  
  color: ${color.text};  
`;  
  
export constInputLabel = styled.Text<InputLabelProps>`  
  font-size: ${(props) => (props.type === "hint" ? "15px" : "18px")};  
  color: ${(props) => {  
    switch (props.type) {  
      case "hint":  
        return color.text;  
      case "error":  
        return color.error.error;  
      default:  
        return color.text;  
    }  
  }};  
  margin-bottom: ${(props) => (props.type === "hint" ? "0px" : "10px")};  
  font-weight: ${(props) => (props.type === "hint" ? "normal" : "500")};  
  text-align: ${(props) => (props.type === "hint" ? "center" : "left")};  
  line-height: ${(props) => (props.type === "hint" ? "18px" : "20px")};  
`;  
  
export const ErrorText = styled.Text`  
  color: ${color.error.error};  
  font-size: 14px;  
  margin-top: 5px;  
  margin-bottom: 10px;  
`;
```

index.tsx

```
import React, { useState, useEffect } from "react";
import { Keyboard } from "react-native";
import Modal from "../modal";
import { ModalBody,InputLabel,InputContainer,ErrorText } from "../style";
import { validateEmail } from "../../utils/validate";
import CustomInput from "packages/ui/components/CustomInput";

interface EmailChangeModalProps {
  visible: boolean;
  onClose: () => void;
  onEmailSubmit?: (email: string) => void;
  currentEmail?: string;
}

const EmailChangeModal: React.FC<EmailChangeModalProps> = ({  
  visible,  
  onClose,  
  onEmailSubmit,  
  currentEmail = "",  
}) => {  
  const [email, setEmail] = useState(currentEmail);  
  const [error, setError] = useState("");  
  const [isLoading, setIsLoading] = useState(false);  
  
  useEffect(() => {  
    if (visible) {  
      setEmail(currentEmail);  
      setError("");  
    }  
  }, [visible, currentEmail]);  
  
  const handleSubmit = async (): Promise<void> => {  
    Keyboard.dismiss();  
  
    if (!email.trim()) {  
      setError("Por favor, insira um endereço de email");  
      return;  
    }  
  
    if (!validateEmail(email)) {  
      setError("Por favor, insira um endereço de email válido");  
      return;  
    }  
  };  
}
```

```
}

if (email.trim().toLowerCase() === currentEmail.toLowerCase()) {
  setError("Este já é o seu email atual");
  return;
}

setError("");
setIsLoading(true);

try {
  await onEmailSubmit?(email.trim());
} catch (error) {
  setError("Erro ao enviar solicitação. Tente novamente.");
} finally {
  setLoading(false);
}
};

const handleClose = (): void => {
  setEmail(currentEmail);
  setError("");
  Keyboard.dismiss();
  onClose();
};

return (
<Modal
  visible={visible}
  onClose={handleClose}
  primaryButtonText={isLoading ? "Enviando..." : "Enviar Confirmação"}
  secondaryButtonText="Cancelar"
  onPrimaryPress={handleSubmit}
  onSecondaryPress={handleClose}
  heightPercentage={45}
  animationType="fade"
>
<ModalBody>
  <InputLabel>
    Para qual endereço de email você deseja alterar?
  </InputLabel>

  <InputContainer>
    <CustomInput
```

```

placeholder="Email"
value={email}
onChangeText={(text: string) => {
  setEmail(text);
  if (error) setError("");
}}
keyboardType="email-address"
autoCapitalize="none"
/>
</InputContainer>

{error ? (
  <ErrorText>{error}</ErrorText>
) : (
  <InputLabel type="hint">
    Enviaremos um email de verificação para confirmar que este email
    pertence a você.
  </InputLabel>
)
}
</ModalBody>
</Modal>
);
};

export default EmailChangeModal;

```

index.tsx

```

import { useState } from "react";
import { Modal, TouchableWithoutFeedback } from "react-native";
import CustomInput from "@hisius/ui/components/CustomInput";
import CustomButton from "@hisius/ui/components/Button";
import { Feather } from "@expo/vector-icons";
import { IPatient } from "../../../../../packages/interfaces/src";
import * as S from "./style";
import CustomPicker from "../../components/customPicker";
import { useFormErrors } from "../../hooks/FormErrors";

type Props = {
  visible: boolean;
  color?: { text: string };
  onSubmit?: (
    data: Pick<
      IPatient,

```

```

    "birthDate" | "cpf" | "gender" | "phone" | "motherName" | "cnsNumber"
  >
) => void;
errors?: Record<string, string>;
};

type GenderType = "" | "MASCULINO" | "FEMININO";

export default function CadastroPopup({
  visible,
  onSubmit,
  color,
  errors,
}: Props) {
  const [birthdate, setBirthdate] = useState("");
  const [cpf, setCpf] = useState("");
  const [gender, setGender] = useState<GenderType>("");
  const [phone, setPhone] = useState("");
  const [icePhone, setIcePhone] = useState("");
  const [motherName, setMotherName] = useState("");
  const [cnsNumber, setCnsNumber] = useState("");

  const { clearFieldError } = useFormErrors();

  const formatDateToISO = (dateString: string): string => {
    const numbers = dateString.replace(/\D/g, "");

    if (numbers.length <= 2) {
      return numbers;
    } else if (numbers.length <= 4) {
      return `${numbers.slice(0, 2)}/${numbers.slice(2)}`;
    } else {
      return `${numbers.slice(0, 2)}/${numbers.slice(2, 4)}/${numbers.slice(4, 8)}`;
    }
  };

  const convertToISO = (dateString: string): string => {
    const parts = dateString.split("/");
    if (parts.length === 3 && parts[2].length === 4) {
      return `${parts[2]}-${parts[1]}-${parts[0]}`;
    }
    return dateString;
  };
}

```

```

const handleDateChange = (text: string) => {
  const formattedDate = formatDateToISO(text);
  setBirthDate(formattedDate);
  if (errors?.birthdate) clearFieldError("birthdate");
};

const handleCpfChange = (text: string) => {
  setCpf(text);
  if (errors?.cpf) clearFieldError("cpf");
};

const handleGenderChange = (value: GenderType) => {
  setGender(value);
  if (errors?.gender) clearFieldError("gender");
};

const handlePhoneChange = (text: string) => {
  setPhone(text);
  if (errors?.phone) clearFieldError("phone");
};

const handleIcePhoneChange = (text: string) => {
  setIcePhone(text);
  if (errors?.icePhone) clearFieldError("icePhone");
};

const handleMotherNameChange = (text: string) => {
  setMotherName(text);
  if (errors?.motherName) clearFieldError("motherName");
};

const handleCnsNumberChange = (text: string) => {
  setCnsNumber(text);
  if (errors?.cnsNumber) clearFieldError("cnsNumber");
};

const handleSubmit = () => {
  const isoDate = convertToISO(birthDate);

  const submitData = {
    birthDate: isoDate,
    cpf,
    gender: gender as "MASCULINO" | "FEMININO",
    phone,
  }
};

```

```

icePhone,
motherName,
cnsNumber,
};

if(gender) {
  onSubmit?.(submitData);
}
};

const validateDate = (dateString: string): boolean => {
  if(dateString.length !== 10) return false;

  const parts = dateString.split("/");
  if(parts.length !== 3) return false;

  const day = parseInt(parts[0], 10);
  const month = parseInt(parts[1], 10);
  const year = parseInt(parts[2], 10);

  if(month < 1 || month > 12) return false;
  if(day < 1 || day > 31) return false;
  if(year < 1900 || year > new Date().getFullYear()) return false;

  return true;
};

const isFormValid =
  validateDate(birthDate) &&
  cpf &&
  gender &&
  phone &&
  motherName &&
  cnsNumber;

return (
  <Modal visible={visible} transparent animationType="fade">
    <TouchableWithoutFeedback>
      <S.Overlay>
        <S.PopupContainer>
          <S.Title>
            Termine de cadastrar seus dados para seguir para a fila
          </S.Title>
        
```

```
<S.InputsContainer>
  <CustomInput
    placeholder="Data de nascimento"
    value={birthdate}
    onChangeText={handleDateChange}
    icon={<Feather name="calendar" size={15} color={color?.text} />}
    keyboardType="numeric"
    maxLength={10}
    error={errors?.birthdate}
  />

  <CustomInput
    placeholder="CPF"
    value={cpf}
    onChangeText={handleCpfChange}
    icon={
      <Feather name="credit-card" size={15} color={color?.text} />
    }
    error={errors?.cpf}
  />

  <CustomPicker
    value={gender}
    onValueChange={handleGenderChange}
    options={[
      { label: "Masculino", value: "MASCULINO" },
      { label: "Feminino", value: "FEMININO" },
    ]}
    placeholder="Sexo"
    icon={<Feather name="user" size={16} color={color?.text} />}
    error={errors?.gender}
  />

  <CustomInput
    placeholder="Telefone"
    value={phone}
    onChangeText={handlePhoneChange}
    icon={<Feather name="phone" size={15} color={color?.text} />}
    error={errors?.phone}
  />

  <CustomInput
    placeholder="Telefone de Emergência"
    value={icePhone}
```

```

        onChangeText={handleIcePhoneChange}
        icon={<Feather name="phone" size={15} color={color?.text} />}
        error={errors?.icePhone}
    />

    <CustomInput
        placeholder="Nome da Mãe"
        value={motherName}
        onChangeText={handleMotherNameChange}
        icon={<Feather name="users" size={15} color={color?.text} />}
        error={errors?.motherName}
    />

    <CustomInput
        placeholder="CNS"
        value={cnsNumber}
        onChangeText={handleCnsNumberChange}
        icon={
            <Feather name="clipboard" size={15} color={color?.text} />
        }
        error={errors?.cnsNumber}
    />
</S.InputsContainer>

<S.ButtonContainer>
    <CustomButton
        title="Cadastrar"
        onPress={handleSubmit}
        disabled={!isValid}
    />
</S.ButtonContainer>
</S.PopupContainer>
</S.Overlay>
</TouchableWithoutFeedback>
</Modal>
);
}

```

style.tsx

```
import styled from "styled-components/native";
```

```
export const Overlay = styled.View`  
flex: 1;
```

```
background-color: rgba(0, 0, 0, 0.5);
justify-content: center;
align-items: center;
padding: 20px;
';

export const PopupContainer = styled.View`  

  background-color: white;  

  border-radius: 12px;  

  padding: 24px;  

  width: 100%;  

  max-width: 400px;  

`;

export const Title = styled.Text`  

  font-size: 18px;  

  font-weight: bold;  

  text-align: center;  

  margin-bottom: 20px;  

  color: #333;  

`;

export const InputsContainer = styled.View`  

  margin-bottom: 20px;  

`;

export const ButtonContainer = styled.View`  

  margin-top: 10px;  

`;

export const GenderFieldContainer = styled.View`  

  position: relative;  

  margin-top: 8px;  

  margin-bottom: 8px;  

`;

export const GenderInputButton = styled.TouchableOpacity<{ active?: boolean }>`  

  background-color: white;  

  border-radius: 8px;  

  border-width: 1px;  

  border-color: ${props => props.active ? '#007AFF' : '#ddd'};  

  padding: 15px;  

  height: 50px;  

  justify-content: center;
```

```
`;  
  
export const GenderInputContent = styled.View`  
  flex-direction: row;  
  justify-content: space-between;  
  align-items: center;  
`;  
  
export const GenderText = styled.Text<{ selected: boolean }>`  
  font-size: 16px;  
  color: ${props => props.selected ? '#000' : '#999'};  
`;  
  
export const GenderOptionsContainer = styled.View`  
  position: absolute;  
  top: 100%;  
  left: 0;  
  right: 0;  
  background-color: white;  
  border-radius: 8px;  
  border-width: 1px;  
  border-color: #ddd;  
  margin-top: 4px;  
  box-shadow: 0px 2px 3px rgba(0, 0, 0, 0.1);  
  z-index: 1000;  
`;  
  
export const GenderOption = styled.TouchableOpacity<{ first?: boolean; last?: boolean }>`  
  padding: 15px;  
  border-bottom-width: ${props => props.last ? 0 : 1}px;  
  border-bottom-color: #f0f0f0;  
  ${props => props.first && `  
    border-top-left-radius: 8px;  
    border-top-right-radius: 8px;  
  `}  
  
  ${props => props.last && `  
    border-bottom-left-radius: 8px;  
    border-bottom-right-radius: 8px;  
  `}  
`;  
  
export const GenderOptionText = styled.Text`  
  font-size: 16px;
```

```

color: #333;
';



## index.tsx


import React from "react";
import ModalContainer from "../modalContainer";
import {
  ModalBody,
  ModalFooter,
  Button,
  ButtonText,
} from "../style";

interface ModalProps {
  visible: boolean;
  onClose: () => void;
  children: React.ReactNode;
  showFooter?: boolean;
  primaryButtonText?: string;
  secondaryButtonText?: string;
  onPrimaryPress?: () => void;
  onSecondaryPress?: () => void;
  heightPercentage?: number;
  animationType?: "none" | "slide" | "fade";
}

const Modal: React.FC<ModalProps> = ({  

  visible,  

  onClose,  

  children,  

  showFooter = true,  

  primaryButtonText = "OK",  

  secondaryButtonText = "Cancelar",  

  onPrimaryPress,  

  onSecondaryPress,  

  heightPercentage = 80,  

  animationType = "slide",  

}) => {  

  const handlePrimaryPress = (): void => {  

    onPrimaryPress?.();  

  };  

  const handleSecondaryPress = (): void => {  

    onSecondaryPress?.();  

  };
}

```

```

    onClose?.());
};

return (
  <ModalContainer
    visible={visible}
    onClose={onClose}
    heightPercentage={heightPercentage}
    animationType={animationType}
  >
  <ModalBody>{children}</ModalBody>

  {showFooter && (
    <ModalFooter>
      <Button onPress={handleSecondaryPress}>
        <ButtonText>{secondaryButtonText}</ButtonText>
      </Button>
      <Button primary onPress={handlePrimaryPress}>
        <ButtonText primary>{primaryButtonText}</ButtonText>
      </Button>
    </ModalFooter>
  )}
</ModalContainer>
);
};

export default Modal;

```

index.tsx

```

import React from "react";
import {
  Modal as RNModal,
  Pressable,
  Keyboard,
  Dimensions,
} from "react-native";
import { ModalOverlay, ModalContent } from "../style";

const { height } = Dimensions.get("window");

interface ModalContainerProps {
  visible?: boolean;
  onClose?: () => void;

```

```

children: React.ReactNode;
heightPercentage?: number;
animationType?: "none" | "slide" | "fade";
transparent?: boolean;
}

const ModalContainer: React.FC<ModalContainerProps> = ({

  visible = false,
  onClose,
  children,
  heightPercentage = 80,
  animationType = "slide",
  transparent = true,
}) => {

  const handleBackdropPress = (): void => {
    Keyboard.dismiss();
    onClose?.();
  };

  const handleContentPress = (event: any): void => {
    event.stopPropagation();
  };

  return (
    <RNModal
      visible={visible}
      animationType={animationType}
      transparent={transparent}
      onRequestClose={onClose}
    >
      <Pressable onPress={handleBackdropPress} style={{ flex: 1 }}>
        <ModalOverlay>
          <Pressable onPress={handleContentPress}>
            <ModalContent height={(height * heightPercentage) / 100}>
              {children}
            </ModalContent>
          </Pressable>
        </ModalOverlay>
      </Pressable>
    </RNModal>
  );
};

export default ModalContainer;

```

index.tsx

```

import React, { useEffect, useState } from "react";
import {
  Alert,
  View,
  ScrollView,
  KeyboardAvoidingView,
  Platform,
} from "react-native";
import * as S from "./style";

import { Auth, getToken, saveToken, saveUser } from "@hisius/services";
import CustomInput from "@hisius/ui/components/CustomInput";
import CustomButton from "@hisius/ui/components/Button";
import { NavigationProp, useNavigation } from "@react-navigation/native";
import { RootStackParamList } from "apps/mobile/navigation/types";
import { Feather } from "@expo/vector-icons";
import { color } from "@hisius/ui/theme/colors";
import { useFormErrors } from "../../hooks/FormErrors";

export default function LoginRegister() {
  const navigation = useNavigation<NavigationProp<RootStackParamList>>();

  const [mode, setMode] = useState<"login" | "register">("login");

  const [email, setEmail] = useState("");
  const [password, setPassword] = useState("");
  const [confirmPassword, setConfirmPassword] = useState("");
  const [name, setName] = useState("");
  const AuthService = new Auth();

  const [loading, setLoading] = useState(false);

  const { errors, clearFieldError, clearAllErrors, handleApiErrors } =
    useFormErrors();

  useEffect(() => {
    const checkToken = async () => {
      const token = await getToken();
      if (token) {
        navigation.navigate("Home");
      }
    }
  })
}

```

```
};

checkToken();
}, []);

const handleSubmit = async () => {
try {
  clearAllErrors();

  setLoading(true);

  if (mode === "login") {
    const data = await AuthService.Login({ email, password });
    const { accessToken, ...rest } = data;
    saveToken(accessToken);
    saveUser(rest);

    navigation.navigate("Home");
    return;
  }

  const response = await AuthService.register({
    name,
    email,
    password,
    confirmPassword,
  });

  const loginData = await AuthService.Login({ email, password });
  const { accessToken, ...rest } = loginData;
  saveToken(accessToken);
  saveUser(rest);

  navigation.navigate("Home");
} catch (error: any) {
  if (error.response?.data) {
    handleApiErrors(error.response.data);
  } else {
    Alert.alert("Erro", "Falha ao processar a ação.");
  }
} finally {
  setLoading(false);
}
};
```

```

const handleEmailChange = (text: string) => {
  setEmail(text);
  if (errors.email) clearFieldError("email");
};

const handlePasswordChange = (text: string) => {
  setPassword(text);
  if (errors.password) clearFieldError("password");
};

const handleConfirmPasswordChange = (text: string) => {
  setConfirmPassword(text);
  if (errors.confirmPassword) clearFieldError("confirmPassword");
};

const handleNameChange = (text: string) => {
  setName(text);
  if (errors.name) clearFieldError("name");
};

const handleModeChange = (newMode: "login" | "register") => {
  setMode(newMode);
  clearAllErrors();
  if (newMode === "login") {
    setConfirmPassword("");
    setName("");
  }
};

return (
  <KeyboardAvoidingView
    style={{ flex: 1 }}
    behavior={Platform.OS === "ios" ? "padding" : "height"}
  >
  <ScrollView
    contentContainerStyle={{ flexGrow: 1 }}
    keyboardShouldPersistTaps="handled"
  >
    <S.Container>
      <S.Title>HISIUS</S.Title>

      <View style={{ flex: 1, width: "100%" }}>
        <S.TabContainer>
          <S.TabButton>

```

```

active={mode === "login"}
onPress={() => handleModeChange("login")}
>
<S.TabText>Entrar</S.TabText>
{mode === "login" && <S.ActiveBar />}
</S.TabButton>

<S.TabButton
active={mode === "register"}
onPress={() => handleModeChange("register")}
>
<S.TabText>Registrar</S.TabText>
{mode === "register" && <S.ActiveBar />}
</S.TabButton>
</S.TabContainer>

<S.InnerContainer>
<S.InputContainer>
{mode === "register" && (
<CustomInput
placeholder="Nome"
value={name}
inputId="name"
icon={<Feather name="user" size={15} color={color.text} />}
onChangeText={handleNameChange}
error={errors.name}
/>
)}
<CustomInput
placeholder="E-mail"
value={email}
inputId="email"
onChangeText={handleEmailChange}
icon={<Feather name="mail" size={15} color={color.text} />}
keyboardType="email-address"
autoCapitalize="none"
error={errors.email}
/>
<CustomInput
placeholder="Senha"
value={password}
inputId="password"

```

```

icon={<Feather name="lock" size={15} color={color.text} />}
onChangeText={handlePasswordChange}
visibilityOff={
  <Feather name="eye-off" size={15} color={color.text} />
}
visibilityOn={
  <Feather name="eye" size={15} color={color.text} />
}
secureTextEntry
error={errors.password}
/>

{mode === "register" && (
<CustomInput
placeholder="Confirmar senha"
value={confirmPassword}
inputId="confirmpassword"
icon={<Feather name="lock" size={15} color={color.text} />}
visibilityOff={
  <Feather name="eye-off" size={15} color={color.text} />
}
visibilityOn={
  <Feather name="eye" size={15} color={color.text} />
}
onChangeText={handleConfirmPasswordChange}
secureTextEntry
error={errors.confirmPassword}
/>
)}
</S.InputContainer>

<CustomButton
title={mode === "login" ? "Entrar" : "Registrar"}
onPress={handleSubmit}
disabled={loading}
/>
</S.InnerContainer>
</View>
</S.Container>
</ScrollView>
</KeyboardAvoidingView>
);
}

```

style.tsx

```
import { color } from "@hisius/ui/theme/colors";
import styled from "styled-components/native";
import { GlobalText } from "../../components/globalText";
import { scale } from "../../utils/scale";

export const Container = styled.View`  
  padding: 32px;  
  display: flex;  
  flex-direction: column;  
  gap: 5rem;  
  height: 100%;  
  background-color: ${color.background};  
`;

export const Title = styled(GlobalText)`  
  font-size: ${scale(32)};  
  letter-spacing: 0.5rem;  
  text-align: center;  
  margin: 80px 0;  
  font-weight: 700;  
`;

export const InnerContainer = styled.View`  
  padding: 20px;  
  background-color: ${color.background};  
  border-top-width: 1px;  
  border-top-color: rgba(0, 0, 0, 0.1);  
`;

export const TabContainer = styled.View`  
  flex-direction: row;  
  justify-content: center;  
  margin-bottom: 15px;  
  gap: 40px;  
  position: relative;  
`;

export const TabButton = styled.TouchableOpacity.attrs({  
  activeOpacity: 0.7,  
})<{ active: boolean }>`  
  align-items: center;  
  width: 30%;
```

```
padding-bottom: 8px;
';

export const TabText = styled(GlobalText)`
  font-size: 18px;
';

export const ActiveBar = styled.View`
  width: 100%;
  height: 3px;
  background-color: ${color.secondary};
  border-radius: 2px;
  margin-top: 4px;
';

export const InputContainer = styled.View`
  width: 100%;
  gap: 16px;
  margin-bottom: 24px;
';

export const Button = styled.TouchableOpacity`
  width: 100%;
  background-color: ${color.secondary};
  padding: 14px;
  border-radius: 8px;
  align-items: center;
  transform: scale(1);
  transition: transform 0.2s ease;

  ${(props) =>
    props.disabled &&
    `
      opacity: 0.6;
    `}
`;

  ${(props) =>
    !props.disabled &&
    `
      transform: scale(0.98);
    `}
';

export const ButtonText = styled(GlobalText)`
```

```
font-size: 18px;
font-weight: bold;
';
```

index.tsx

```
import React, { useEffect, useRef, useState } from "react";
import { Alert, TextInput } from "react-native";
import { NavigationProp, useNavigation } from "@react-navigation/native";
import * as S from "./style";
import CustomButton from "@hisius/ui/components/Button";
import { Patient, Queue } from "@hisius/services/src";
import Header from "../../components/header";
import { RootStackParamList } from "apps/mobile/navigation/types";
import CadastroPopup from "../../popups/checkinData";
import { useFormErrors } from "../../hooks/FormErrors";
import { useNotification } from "../../components/notification/context";

const CODE_LENGTH = 6;

export default function HomeScreen() {
  const [code, setCode] = useState(Array(CODE_LENGTH).fill(""));
  const [loading, setLoading] = useState(false);
  const [showCadastroPopup, setShowCadastroPopup] = useState(false);
  const inputsRef = useRef(
    Array(CODE_LENGTH)
      .fill(0)
      .map(() => React.createRef<TextInput>())
  );
  const navigation = useNavigation<NavigationProp<RootStackParamList>>();
  const queue = new Queue();
  const patient = new Patient();
  const { addNotification } = useNotification();
  const { errors, clearAllErrors, handleApiErrors } = useFormErrors();

  useEffect(() => {
    patient.getQueueInfo().then((info) => {
      if (info) navigation.navigate("Queue");
    });
  }, []);

  const handleCodeChange = (text: string, index: number) => {
    const num = text.replace(/\^0-9/g, "");
    const newCode = [...code];
    newCode[index] = num;
    setCode(newCode);
  };
}
```

```

if (num.length > 1) {
  const digits = num.split("").slice(0, CODE_LENGTH);
  const filledCode = [
    ...digits,
    ...Array(CODE_LENGTH - digits.length).fill(""),
  ];
  setCode(filledCode);
  inputsRef.current[
    Math.min(digits.length, CODE_LENGTH - 1)
  ].current?.focus();
  return;
}

newCode[index] = num;
setCode(newCode);

if (num && index < CODE_LENGTH - 1)
  inputsRef.current[index + 1].current?.focus();
};

const handleKeyPress = (e: any, index: number) => {
  if (e.nativeEvent.key === "Backspace" && !code[index] && index > 0)
    inputsRef.current[index - 1].current?.focus();
};

const handleCadastroSubmit = async (patientData: any) => {
  try {
    console.log("Dados do paciente:", patientData);
    await patient.createProfile(patientData);
    setShowCadastroPopup(false);
    addNotification("Dados cadastrados com sucesso!", "success");
  } catch (error: any) {
    console.error("Erro no cadastro:", error);
    if (error.response?.data) {
      handleApiErrors(error.response.data);
    } else {
      addNotification("Não foi possível completar o cadastro.", "error");
    }
  }
};

const handleJoin = async () => {
  if (code.some((d) => !d)) {

```

```
addNotification("Preencha todos os 6 dígitos.", "warning");
return;
}

 setLoading(true);
clearAllErrors();
try {
  const success = await queue.joinQueue(code.join(""));
  if (success) navigation.navigate("Queue");
} catch (error: any) {
  if (
    error.response?.data?.message?.includes(
      "Perfil de paciente não encontrado para este usuário."
    )
  ) {
    setShowCadastroPopup(true);
    return;
  }

  if (error.response?.data) {
    handleApiErrors(error.response.data);
  } else {
    addNotification("Não foi possível entrar na fila.", "error");
  }
} finally {
  setLoading(false);
}
};

return (
<S.Container>
  <Header
    softwareName="Hisius"
    onProfilePress={() => navigation.navigate("Profile")}
  />
  <S.ContentContainer>
    <S.HeaderContainer>
      <S.Title>ESTÁ NO HOSPITAL?</S.Title>
      <S.Subtitle>Digite o código de acesso</S.Subtitle>
    </S.HeaderContainer>
    <S.CodeContainer>
      {code.map((value, i) => (
        <S.CodeInput
          key={i}

```

```

    ref={inputsRef.current[i]}
    placeholder="0"
    keyboardType="number-pad"
    maxLength={1}
    value={value}
    onChangeText={(t) => handleCodeChange(t, i)}
    onKeyPress={(e) => handleKeyPress(e, i)}
    selectTextOnFocus={true}
  />
))
)
</S.CodeContainer>
<S.ButtonContainer>
<CustomButton
  title={loading ? "Entrando..." : "Entrar na Fila"}
  onPress={handleJoin}
  disabled={loading || code.some((d) => !d)}
/>
</S.ButtonContainer>
</S.ContentContainer>

<CadastroPopup
  visible={showCadastroPopup}
  onSubmit={handleCadastroSubmit}
  color={{ text: "#000" }}
  errors={errors}
/>
</S.Container>
);
}

```

style.tsx

```

import styled from "styled-components/native";
import { CodeInputBase } from "../../components/codeInput";
import { GlobalText } from "../../components/globalText";
import { scale } from "../../utils/scale";
import { color } from "@hisius/ui/theme/colors";

export const Container = styled.View`
  flex: 1;
  background-color: ${color.background};
  align-items: center;
  padding: 40px 20px;
  gap: 40px;

```

```
`;  
  
export const ContentContainer = styled.View`  
  flex-direction: column;  
  align-items: center;  
  margin-top: 5rem;  
  height: 100%;  
  gap: 5rem;  
`;  
  
export const HeaderContainer = styled.View`  
  align-items: center;  
  margin-top: 40px;  
`;  
  
export const Title = styled(GlobalText)`  
  font-size: ${scale(20)};  
  font-weight: 600;  
  margin-bottom: 8px;  
`;  
  
export const Subtitle = styled(GlobalText)`  
  font-size: ${scale(14)};  
  text-align: center;  
`;  
  
export const CodeContainer = styled.View`  
  flex-direction: row;  
  justify-content: center;  
  gap: 12px;  
  margin: 20px 0;  
`;  
  
export const CodeInput = styled(CodeInputBase)`  
  width: 50px;  
  height: 60px;  
  border-radius: 12px;  
  border: 2px solid ${color.gray};  
  color: ${color.text};  
  background: ${color.card};  
  font-size: ${scale(20)};  
  text-align: center;  
  font-family: "Montserrat";  
  font-weight: 500;
```

```

';
export const ButtonContainer = styled.View`  

  width: 100%;  

  margin-top: 20px;  

`;

```

index.tsx

```

import React, { useState, useEffect } from "react";
import { View, Alert } from "react-native";
import CustomInput from "@hisius/ui/components/CustomInput";
import CustomButton from "@hisius/ui/components/Button";
import { Auth, getUser, logout, Patient } from "@hisius/services/src";
import { color } from "@hisius/ui/theme/colors";
import { Feather } from "@expo/vector-icons";
import { NavigationProp, useNavigation } from "@react-navigation/native";
import { IPatient } from "packages/interfaces/src";
import CustomPicker from "../../components/customPicker";
import { RootStackParamList } from "apps/mobile/navigation/types";
import * as S from "./style";
import EmailChangeModal from "../../popups/changeEmail";
import { useFormErrors } from "../../hooks/FormErrors";
import { useNotification } from "../../components/notification/context";

export function Profile() {
  const patientInstance = new Patient();
  const AuthService = new Auth();
  const navigation = useNavigation<NavigationProp<RootStackParamList>>();

  const [name, setName] = useState("");
  const [cpf, setCpf] = useState("");
  const [gender, setGender] = useState<"" | "MASCULINO" | "FEMININO">("");
  const [birthDate, setBirthDate] = useState("");
  const [cnsNumber, setCnsNumber] = useState("");
  const [email, setEmail] = useState("");
  const [phone, setPhone] = useState("");
  const [motherName, setMotherName] = useState("");

  const [initialData, setInitialData] = useState<Partial<IPatient>>({});  

  const [isEmailModalVisible, setIsEmailModalVisible] = useState(false);  

  const [loading, setLoading] = useState(false);

  const { addNotification } = useNotification();

```

```

const { errors, clearFieldError, clearAllErrors, handleApiErrors } =
useFormErrors();

const openEmailModal = () => setIsEmailModalVisible(true);
const closeEmailModal = () => setIsEmailModalVisible(false);

const formatDateFromISO = (isoDate: string): string => {
  if (!isoDate) return "";
  if (isoDate.includes("/")) return isoDate;

  const parts = isoDate.split("-");
  if (parts.length === 3) {
    return `${parts[2]}/${parts[1]}/${parts[0]}`;
  }
  return isoDate;
};

const formatDateToISO = (dateString: string): string => {
  const numbers = dateString.replace(/\D/g, "");

  if (numbers.length <= 2) {
    return numbers;
  } else if (numbers.length <= 4) {
    return `${numbers.slice(0, 2)}/${numbers.slice(2)}`;
  } else {
    return `${numbers.slice(0, 2)}/${numbers.slice(2, 4)}/${numbers.slice(4, 8)}`;
  }
};

const convertToISO = (dateString: string): string => {
  const parts = dateString.split("/");
  if (parts.length === 3 && parts[2].length === 4) {
    return `${parts[2]}-${parts[1]}-${parts[0]}`;
  }
  return dateString;
};

const handleEmailSubmit = async (newEmail: string) => {
  try {
    setLoading(true);
    await AuthService.changeEmail(newEmail);
    addNotification("Link de confirmação enviado!");
    closeEmailModal();
  }
};

```

```
    } catch (err: any) {
      if (err.response?.data) {
        handleApiErrors(err.response.data);
      } else {
        addNotification(
          "Erro ao enviar código de verificação. Tente novamente.",
          "error"
        );
      }
    } finally {
  setLoading(false);
}
};

const handleBack = () => {
  navigation.goBack();
};

const handleLogout = async () => {
  await logout();
  navigation.reset({
    index: 0,
    routes: [{ name: "Login" }],
  });
};

const handleChangePassword = async () => {
  try {
    const user = JSON.parse(await getUser());
    await AuthService.changePass(user.email);
    addNotification(
      "Instruções para alteração de senha enviadas para seu e-mail."
    );
  } catch (err: any) {
    if (err.response?.data) {
      handleApiErrors(err.response.data);
    } else {
      addNotification("Falha ao solicitar alteração de senha.", "error");
    }
  }
};

const handleChangeEmail = () => {
  openEmailModal();
```

```

};

const handleSave = async () => {
  try {
    setLoading(true);
    clearAllErrors();

    const isoBirthDate = convertToISO(birthDate);

    const updatedPatient: IPatient = {
      name,
      cpf,
      gender: gender as "MASCULINO" | "FEMININO",
      birthDate: isoBirthDate,
      cnsNumber,
      email,
      phone,
      motherName,
    };
    const success = await patientInstance.updateProfile(updatedPatient);
    if (success) {
      addNotification("Perfil atualizado com sucesso!");
      setInitialData({
        ...updatedPatient,
        birthDate: birthDate,
      });
    } else {
      addNotification("Erro ao atualizar perfil.", "error");
    }
  } catch (err: any) {
    if (err.response?.data) {
      handleApiErrors(err.response.data);
    } else {
      addNotification("Erro ao atualizar perfil.", "error");
    }
  } finally {
    setLoading(false);
  }
};

const handleBirthDateChange = (text: string) => {
  const formattedDate = formatDateToISO(text);
  setBirthDate(formattedDate);
  if (errors.birthDate) clearFieldError("birthDate");
}

```

```
};

const handleNameChange = (text: string) => {
  setName(text);
  if (errors.name) clearFieldError("name");
};

const handleCpfChange = (text: string) => {
  setCpf(text);
  if (errors.cpf) clearFieldError("cpf");
};

const handleEmailChange = (text: string) => {
  setEmail(text);
  if (errors.email) clearFieldError("email");
};

const handlePhoneChange = (text: string) => {
  setPhone(text);
  if (errors.phone) clearFieldError("phone");
};

const handleCnsNumberChange = (text: string) => {
  setCnsNumber(text);
  if (errors.cnsNumber) clearFieldError("cnsNumber");
};

const handleMotherNameChange = (text: string) => {
  setMotherName(text);
  if (errors.motherName) clearFieldError("motherName");
};

const handleGenderChange = (value: "" | "MASCULINO" | "FEMININO") => {
  setGender(value);
  if (errors.gender) clearFieldError("gender");
};

const hasValidChanges = () => {
  const currentData = {
    name,
    cpf,
    gender,
    birthDate,
    cnsNumber,
```

```
email,  
phone,  
motherName,  
};  
  
const hasChanges = Object.keys(currentData).some((key) => {  
  const currentValue = currentData[key as keyof typeof currentData];  
  const initialValue = initialData[key as keyof typeof initialData];  
  return currentValue !== initialValue;  
});  
  
const hasEmptyFields = !name || !cpf || !birthDate || !email;  
  
return hasChanges && !hasEmptyFields;  
};  
  
useEffect(() => {  
  patientInstance  
    .getProfile()  
    .then((data) => {  
      if (!data) return;  
  
      const patientData = {  
        name: data.name ?? "",  
        cpf: data.cpf ?? "",  
        gender: data.gender as "MASCULINO" | "FEMININO",  
        birthDate: formatDateFromISO(data.birthDate ?? ""),  
        cnsNumber: data.cnsNumber ?? "",  
        email: data.email ?? "",  
        phone: data.phone ?? "",  
        motherName: data.motherName ?? "",  
      };  
  
      setName(patientData.name);  
      setCpf(patientData.cpf);  
      setGender(patientData.gender || "");  
      setBirthDate(patientData.birthDate);  
      setCnsNumber(patientData.cnsNumber);  
      setEmail(patientData.email);  
      setPhone(patientData.phone);  
      setMotherName(patientData.motherName);  
  
      setInitialData(patientData);  
    })
```

```
.catch(() => {
  navigation.goBack();
  addNotification("Erro ao buscar informações do perfil!", "error");
});
}, []);

return (
<S.Container>
<S.Header>
  <S.BackButton onPress={handleBack}>
    <Feather name="arrow-left" size={24} color={color.text} />
  </S.BackButton>

  <S.TitleBox>
    <S.Subtitle>Editar informações</S.Subtitle>
    <S.Title>Meu Perfil</S.Title>
  </S.TitleBox>
</S.Header>

<S.FormContainer>
<S.Section>
  <S.SectionTitle>Informações Pessoais</S.SectionTitle>

  <S.InputGroup>
    <CustomInput
      placeholder="Nome completo"
      value={name}
      onChangeText={handleNameChange}
      icon={<Feather name="user" size={16} color={color.text} />}
      error={errors.name}
    />
  </S.InputGroup>

  <S.InputRow>
    <S.InputColumn>
      <CustomInput
        placeholder="Data de Nascimento"
        value={birthDate}
        onChangeText={handleBirthDateChange}
        icon={<Feather name="calendar" size={16} color={color.text} />}
        error={errors.birthDate}
        keyboardType="numeric"
        maxLength={10}
      />
```

```

</S.InputColumn>

<S.InputColumn>
  <CustomPicker
    value={gender}
    onValueChange={handleGenderChange}
    options={[
      { label: "Masculino", value: "MASCULINO" },
      { label: "Feminino", value: "FEMININO" },
    ]}
    placeholder="Sexo"
    icon={<Feather name="user" size={16} color={color.text} />}
    error={errors.gender}
  />
</S.InputColumn>
</S.InputRow>

<S.InputGroup>
  <CustomInput
    placeholder="Nome da M  e"
    value={motherName}
    onChangeText={handleMotherNameChange}
    icon={<Feather name="users" size={16} color={color.text} />}
    error={errors.motherName}
  />
</S.InputGroup>
</S.Section>

<S.Section>
  <S.SectionTitle>Documentos</S.SectionTitle>

  <S.InputRow>
    <S.InputColumn>
      <CustomInput
        placeholder="CPF"
        value={cpf}
        onChangeText={handleCpfChange}
        icon={
          <Feather name="credit-card" size={16} color={color.text} />
        }
        error={errors.cpf}
      />
    </S.InputColumn>
  
```

```
<S.InputColumn>
  <CustomInput
    placeholder="CNS"
    value={cnsNumber}
    onChangeText={handleCnsNumberChange}
    icon={<Feather name="clipboard" size={16} color={color.text} />}
    error={errors.cnsNumber}
  />
</S.InputColumn>
</S.InputRow>
</S.Section>

<S.Section>
  <S.SectionTitle>Contato</S.SectionTitle>

  <S.InputGroup>
    <CustomInput
      placeholder="Email"
      value={email}
      onChangeText={handleEmailChange}
      disabled
      icon={<Feather name="mail" size={16} color={color.text} />}
      error={errors.email}
    />
  </S.InputGroup>

  <S.InputGroup>
    <CustomInput
      placeholder="Telefone"
      value={phone}
      onChangeText={handlePhoneChange}
      icon={<Feather name="phone" size={16} color={color.text} />}
      error={errors.phone}
    />
  </S.InputGroup>
</S.Section>

<S.Section>
  <S.SectionTitle>Segurança</S.SectionTitle>

  <S.SecurityActions>
    <S.SecurityItem onPress={handleChangeEmail}>
      <S.SecurityIcon>
        <Feather name="mail" size={20} color={color.primary} />
      </S.SecurityIcon>
    </S.SecurityItem>
  </S.SecurityActions>
```

```
</S.SecurityIcon>
<S.SecurityTextContainer>
  <S.SecurityTitle>Alterar Email</S.SecurityTitle>
  <S.SecurityDescription>
    Modifique seu endereço de email
  </S.SecurityDescription>
</S.SecurityTextContainer>
<Feather name="chevron-right" size={20} color={color.text} />
</S.SecurityItem>

<S.SecurityItem onPress={handleChangePassword}>
  <S.SecurityIcon>
    <Feather name="lock" size={20} color={color.primary} />
  </S.SecurityIcon>
  <S.SecurityTextContainer>
    <S.SecurityTitle>Alterar Senha</S.SecurityTitle>
    <S.SecurityDescription>
      Atualize sua senha de acesso
    </S.SecurityDescription>
  </S.SecurityTextContainer>
  <Feather name="chevron-right" size={20} color={color.text} />
</S.SecurityItem>

<S.SecurityItem onPress={handleLogout}>
  <S.SecurityIcon>
    <Feather name="log-out" size={20} color={color.error.error} />
  </S.SecurityIcon>
  <S.SecurityTextContainer>
    <S.SecurityTitle>Logout</S.SecurityTitle>
    <S.SecurityDescription>Saia da sua conta</S.SecurityDescription>
  </S.SecurityTextContainer>
  <Feather name="chevron-right" size={20} color={color.text} />
</S.SecurityItem>
</S.SecurityActions>
</S.Section>

<S.ActionsContainer>
<View>
  <CustomButton
    title="Salvar Alterações"
    onPress={handleSave}
    disabled={!isValidChanges() || loading}
  />
</View>
```

```

    </S.ActionsContainer>
    </S.FormContainer>

    <EmailChangeModal
        visible={isEmailModalVisible}
        onClose={closeEmailModal}
        onEmailSubmit={handleEmailSubmit}
        currentEmail={email}
    />
    </S.Container>
);
}

```

style.tsx

```

import styled from "styled-components/native";
import { View, TouchableOpacity, ScrollView } from "react-native";
import { GlobalText } from "../../components/globalText";
import { color } from "@hisius/ui/theme/colors";

export const Container = styled(ScrollView)` 
flex: 1;
background-color: ${color.background};
padding: 24px;
`;

export const Header = styled(View)` 
flex-direction: row;
align-items: center;
margin-bottom: 40px;
padding-top: 16px;
`;

export const BackButton = styled(TouchableOpacity)` 
margin-right: 20px;
padding: 8px;
background-color: ${color.front};
border-radius: 12px;
`;

export const TitleBox = styled(View)` 
flex: 1;
`;

```

```
export const Title = styled(GlobalText)`  
  font-size: 28px;  
  font-weight: 700;  
  color: ${color.text};  
  letter-spacing: -0.5px;  
`;  
  
export const Subtitle = styled(GlobalText)`  
  font-size: 16px;  
  font-weight: 400;  
  color: ${color.text};  
  margin-bottom: 4px;  
`;  
  
export const FormContainer = styled(View)`  
  gap: 32px;  
`;  
  
export const Section = styled(View)`  
  gap: 16px;  
`;  
  
export const SectionTitle = styled(GlobalText)`  
  font-size: 18px;  
  font-weight: 600;  
  color: ${color.text};  
  margin-bottom: 8px;  
  padding-left: 12px;  
  border-left-width: 3px;  
  border-left-color: ${color.primary};  
`;  
  
export const InputGroup = styled(View)`  
  width: 100%;  
`;  
  
export const InputRow = styled(View)`  
  flex-direction: row;  
  gap: 16px;  
`;  
  
export const InputColumn = styled(View)`  
  flex: 1;  
`;
```

```

export const ActionsContainer = styled(View)`  

  margin-top: 40px;  

  gap: 24px;  

`;  
  

export const SecurityActions = styled(View)`  

  gap: 12px;  

`;  
  

export const SecurityItem = styled(TouchableOpacity)`  

  flex-direction: row;  

  align-items: center;  

  padding: 16px;  

  background-color: ${color.front};  

  border-radius: 12px;  

  border: 1px solid ${color.gray};  

`;  
  

export const SecurityIcon = styled(View)`  

  margin-right: 12px;  

`;  
  

export const SecurityTextContainer = styled(View)`  

  flex: 1;  

`;  
  

export const SecurityTitle = styled(GlobalText)`  

  font-size: 16px;  

  font-weight: 600;  

  color: ${color.text};  

  margin-bottom: 2px;  

`;  
  

export const SecurityDescription = styled(GlobalText)`  

  font-size: 14px;  

  color: ${color.text};  

`;

```

index.tsx

```

import React, { useEffect, useState, useRef } from "react";
import {
  View,
  TouchableOpacity,

```

```

ScrollView,
RefreshControl,
} from "react-native";
import { NavigationProp, useNavigation } from "@react-navigation/native";
import { createStyles } from "./style";
import { GlobalText as Text } from "../../components/globalText";
import { getToken, getUser, Patient, useSocket } from "@hisius/services";
import { IQueuedInfo } from "@hisius/interfaces/src";
import { RootStackParamList } from "../../../../../navigation/types";
import Header from "../../components/header";
import { Feather } from "@expo/vector-icons";
import { color } from "@hisius/ui/theme/colors";
import { useNotification } from "../../../../../components/notification/context";

export function QueueScreen() {
  const [patient, setPatient] = useState<IQueuedInfo | null>(null);
  const [estimatedWaitingTime, setEstimatedWaitingTime] = useState(0);
  const [refreshing, setRefreshing] = useState(false);
  const [lastUpdate, setLastUpdate] = useState(new Date());
  const [isLoading, setIsLoading] = useState(false);
  const [refreshFlag, setRefreshFlag] = useState(false);
  const [user, setUser] = useState<any>(null);
  const [token, setToken] = useState<string | null>(null);
  const [authLoaded, setAuthLoaded] = useState(false);

  const lastRequestTime = useRef(0);
  const patientInstance = new Patient();
  const navigation = useNavigation<NavigationProp<RootStackParamList>>();

  const MIN_REQUEST_DELAY = 5000;
  const AUTO_UPDATE_INTERVAL = 300000;

  const { addNotification } = useNotification();

  useEffect(() => {
    const loadAuthData = async () => {
      try {
        const [userData, authToken] = await Promise.all([
          JSON.parse(await getUser()),
          getToken(),
        ]);

        setUser(userData);
        setToken(authToken);
      }
    }
  });
}

```

```
    setAuthLoaded(true);

    console.log("Dados de autenticação carregados:", {
      userId: userData?.id,
      hasToken: !!authToken,
    });
  } catch (error) {
    console.error("Erro ao carregar dados de autenticação:", error);
    setAuthLoaded(true);
  }
};

loadAuthData();
}, []);

const { lastNotification } = useSocket(
  authLoaded ? user?.id || null : null,
  authLoaded ? token : null
);

useEffect(() => {
  if (lastNotification) {
    console.log(
      "Notificação recebida, recarregando dados...",
      lastNotification
    );
    fetchPatient();
  }
}, [lastNotification]);

const handleProfile = () => navigation.navigate("Profile");
const handleLeaveQueue = async () => {
  await patientInstance.leaveQueue();
  navigation.navigate("Home");
};

const canMakeRequest = () =>
  Date.now() - lastRequestTime.current >= MIN_REQUEST_DELAY;

const fetchPatient = async () => {
  if (isLoading || !canMakeRequest()) return;

  try {
    setIsLoading(true);
```

```

setRefreshing(true);
lastRequestTime.current = Date.now();
setRefreshFlag((prev) => !prev);

const info = await patientInstance.getQueueInfo();
setPatient(info);
setEstimatedWaitingTime(info.estimatedWaitMinutes ?? 0);
setLastUpdate(new Date());
} catch (err) {
  addNotification("Erro ao buscar informações", "error");
  navigation.navigate("Home");
} finally {
  setIsLoading(false);
  setRefreshing(false);
}
};

useEffect(() => {
  fetchPatient();
  const interval = setInterval(fetchPatient, AUTO_UPDATE_INTERVAL);
  return () => clearInterval(interval);
}, []);

useEffect(() => {
  if (!canMakeRequest()) {
    const timer = setTimeout(
      () => setRefreshFlag((prev) => !prev),
      MIN_REQUEST_DELAY
    );
    return () => clearTimeout(timer);
  }
}, [refreshFlag]);

if (!patient)
  return (
    <View style={{ flex: 1, justifyContent: "center", alignItems: "center" }}>
      <Text>Carregando informações...</Text>
    </View>
  );
}

const styles = createStyles(patient.classification);
const canRefresh = canMakeRequest();
const refreshControl = (
  <RefreshControl

```

```
refreshing={refreshing}
onRefresh={fetchPatient}
enabled={canRefresh}
/>
);

if (patient.roomCalled)
  return (
    <ScrollView style={styles.container} refreshControl={refreshControl}>
      <Header softwareName="Hisius" onProfilePress={handleProfile} />

      <View style={styles.card}>
        <View style={styles.cardContent}>
          <View style={styles.calledRoomSection}>
            <View style={styles.titleRow}>
              <Text style={styles.titleText}>Você foi chamado!</Text>
              <TouchableOpacity
                style={[
                  styles.refreshButton,
                  (!canRefresh || isLoading) && { opacity: 0.5 },
                  { marginLeft: 20 },
                ]}
                onPress={fetchPatient}
                disabled={!canRefresh || isLoading}
              >
                <Feather name="refresh-cw" size={20} color={color.front} />
              </TouchableOpacity>
            </View>
          <Text style={styles.calledRoomText}>
            {patient.roomCalled.toLowerCase().includes("sala")
              ? patient.roomCalled
              : "Sala " + patient.roomCalled}
          </Text>

          <Text style={[styles.instructionsItem]}>
            Dirija-se imediatamente à sala indicada.
          </Text>

          <Text style={styles.lastUpdateText}>
            Última atualização: {lastUpdate.toLocaleTimeString()}
          </Text>
        </View>
      </View>
```

```

        </View>

        <View style={styles.footer}>
            <TouchableOpacity
                style={styles.leaveButton}
                onPress={handleLeaveQueue}
            >
                <Text style={styles.leaveButtonText}>Sair da fila</Text>
            </TouchableOpacity>
        </View>
    </ScrollView>
);

return (
    <ScrollView style={styles.container} refreshControl={refreshControl}>
        <Header softwareName="Hisius" onProfilePress={handleProfile} />
        <View style={styles.card}>
            <View style={stylesCardContent}>
                <View style={styles.infoSection}>
                    <View style={styles.titleRow}>
                        <Text style={styles.titleText}>
                            Fila para{" "}
                        <Text style={styles.titleHighlight}>
                            {patient.classification ? "atendimento" : "triagem"}
                        </Text>
                    </Text>
                    <TouchableOpacity
                        style={[
                            styles.refreshButton,
                            (!canRefresh || isLoading) && { opacity: 0.5 },
                        ]}
                        onPress={fetchPatient}
                        disabled={!canRefresh || isLoading}
                    >
                        <Feather name="refresh-cw" size={20} color={color.front} />
                    </TouchableOpacity>
                </View>
                <View style={styles.infoBlock}>
                    <Text style={styles.infoLabel}>Tempo estimado de espera</Text>
                    <View style={styles.timeRow}>
                        <Text style={styles.timeValue}>
                            {estimatedWaitingTime.toString().padStart(2, "0")}
                        </Text>
                        <Text style={styles.timeUnit}>minutos</Text>
                
```

```

        </View>
    </View>
    <View style={styles.infoBlock}>
        <Text style={styles.infoLabel}>Identificação</Text>
        <Text style={styles.infoValue}>#{patient.id}</Text>
    </View>
    <View style={styles.infoBlock}>
        <Text style={styles.infoLabel}>Classificação de risco</Text>
        <Text style={styles.riskValue}>
            {patient.classification || "Aguardando classificação"}
        </Text>
        <Text style={styles.lastUpdateText}>
            Última atualização: {lastUpdate.toLocaleTimeString()}
        </Text>
    </View>
</View>
<View style={styles.instructionsSection}>
    <Text style={styles.instructionsTitle}>Orientações</Text>
    <Text style={styles.instructionsItem}>
        • Mantenha-se no local indicado
    </Text>
    <Text style={styles.instructionsItem}>
        • Fique atento ao chamado do médico
    </Text>
    <Text style={styles.instructionsItem}>
        • Em caso de piora, procure a recepção
    </Text>
</View>
</View>
</View>
<View style={styles.footer}>
    <TouchableOpacity style={styles.leaveButton} onPress={handleLeaveQueue}>
        <Text style={styles.leaveButtonText}>Sair da fila</Text>
    </TouchableOpacity>
</View>
</ScrollView>
);
}

```

style.tsx

```

import { StyleSheet, Dimensions } from "react-native";
import { moderateScale, scale } from "../../utils/scale";
import { getRiskColor } from "../../utils/riskColor";

```

```
import { ManchesterTriage } from "@hisius/enums";
import { color } from "@hisius/ui/theme/colors";

const { height } = Dimensions.get("window");

const PRIMARY_TEXT = color.text;
const LIGHT_BG = color.background;
const CARD_BG = color.card;
const BORDER_COLOR = color.error.error;
const SECONDARY_TEXT = `${PRIMARY_TEXT}99`;

export const createStyles = (risk: ManchesterTriage) => {
  const riskColor = getRiskColor(risk);

  return StyleSheet.create({
    container: {
      flex: 1,
      padding: scale(16),
      backgroundColor: LIGHT_BG,
    },
    header: {
      paddingHorizontal: scale(16),
      paddingTop: scale(12),
      paddingBottom: scale(8),
      flexDirection: "row",
      alignItems: "center",
      justifyContent: "space-between",
    },
    logo: {
      fontSize: moderateScale(18),
      fontWeight: "800",
      letterSpacing: scale(2),
      color: PRIMARY_TEXT,
    },
    profileButton: {
      paddingHorizontal: scale(14),
      paddingVertical: scale(8),
      borderRadius: scale(10),
      backgroundColor: CARD_BG,
      shadowColor: "#000",
      shadowOpacity: 0.06,
      shadowOffset: { width: 0, height: 1 },
      shadowRadius: 4,
      elevation: 2,
    }
  });
}
```

```
},
profileButtonText: {
  fontSize: moderateScale(12),
  fontWeight: "600",
  color: PRIMARY_TEXT,
},
card: {
  width: "100%",
  maxWidth: scale(320),
  minHeight: height * 0.65,
  borderLeftWidth: 5,
  borderLeftColor: riskColor,
  alignSelf: "center",
  marginVertical: scale(30),
  backgroundColor: CARD_BG,
  borderRadius: scale(12),
  shadowColor: "#000",
  shadowOpacity: 0.08,
  shadowOffset: { width: 0, height: 8 },
  shadowRadius: 12,
  elevation: 4,
},
CardContent: {
  flex: 1,
  paddingHorizontal: scale(20),
  paddingVertical: scale(20),
  gap: scale(16),
},
infoSection: {
  marginBottom: scale(24),
},
titleRow: {
  flexDirection: "row",
  alignItems: "center",
  justifyContent: "space-between",
  marginBottom: scale(20),
  paddingBottom: scale(12),
  borderBottomWidth: 1,
  borderBottomColor: `${PRIMARY_TEXT}10`,
},
refreshButton: {
  backgroundColor: color.primary,
  padding: 12,
  aspectRatio: "1/1",
```

```
borderRadius: "50%",  
alignItems: "center",  
},  
refreshButtonText: {  
  color: "white",  
  fontSize: 16,  
  fontWeight: "600",  
},  
lastUpdateText: {  
  fontSize: 12,  
  color: color.text,  
  textAlign: "center",  
  marginTop: 8,  
},  
titleText: {  
  fontSize: moderateScale(18),  
  fontWeight: "600",  
  color: PRIMARY_TEXT,  
  lineHeight: moderateScale(24),  
},  
titleHighlight: {  
  fontWeight: "700",  
  color: riskColor,  
},  
infoBlock: {  
  alignItems: "flex-start",  
  marginBottom: scale(20),  
},  
infoLabel: {  
  fontSize: moderateScale(11),  
  color: SECONDARY_TEXT,  
  fontWeight: "500",  
  marginBottom: scale(4),  
  textTransform: "uppercase",  
  letterSpacing: scale(0.3),  
},  
infoValue: {  
  fontSize: moderateScale(15),  
  fontWeight: "700",  
  color: PRIMARY_TEXT,  
},  
timeRow: {  
  flexDirection: "row",  
  alignItems: "flex-end",
```

```
},
timeValue: {
  fontSize: moderateScale(32),
  fontWeight: "800",
  color: PRIMARY_TEXT,
  marginRight: scale(6),
},
timeUnit: {
  fontSize: moderateScale(13),
  fontWeight: "600",
  color: SECONDARY_TEXT,
  marginBottom: scale(4),
},
riskValue: {
  fontSize: moderateScale(16),
  fontWeight: "800",
  color: riskColor,
  textTransform: "uppercase",
  letterSpacing: scale(0.3),
},
instructionsSection: {
  marginTop: scale(4),
  padding: scale(16),
  backgroundColor: `${PRIMARY_TEXT}05`,
  borderRadius: scale(8),
},
instructionsTitle: {
  fontSize: moderateScale(13),
  fontWeight: "700",
  color: PRIMARY_TEXT,
  marginBottom: scale(12),
},
instructionsItem: {
  fontSize: moderateScale(12),
  lineHeight: moderateScale(18),
  color: SECONDARY_TEXT,
  marginBottom: scale(6),
},
footer: {
  paddingBottom: scale(20),
  alignItems: "center",
},
leaveButton: {
  paddingHorizontal: scale(40),
```

```

paddingVertical: scale(12),
borderRadius: scale(8),
borderWidth: 1.5,
borderColor: BORDER_COLOR,
backgroundColor: "transparent",
},
leaveButtonText: {
fontSize: moderateScale(13),
color: color.error.error,
fontWeight: "700",
},
calledRoomSection: {
alignItems: "center",
justifyContent: "center",
flex: 1,
},
calledRoomText: {
fontSize: moderateScale(26),
fontWeight: "800",
color: riskColor,
textAlign: "center",
marginVertical: scale(16),
},
});
};

```

index.tsx

```

import { View, Text, Image, StyleSheet } from "react-native";
import React, { useEffect } from 'react';
import { useNavigation } from "@react-navigation/native";

export default function Splash() {

  const navigation = useNavigation();

  useEffect(() => {
    const timer = setTimeout(() => {
      navigation.navigate('Login' as never);
    }, 2000);

    return () => clearTimeout(timer);
  }, [navigation]);
}

```

```
return (
  <View style={styles.container}>
    <Text style={styles.title}>HISIUS</Text>
    <Text style={styles.subtitle}>Pronto para agilizar o seu atendimento</Text>

    <View style={styles.imageSpace}>

      <Image source={require('../assets/vectorArt.png')} style={styles.image} />
    </View>
  </View>
);

}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    padding: 20,
    backgroundColor: '#fff'
  },
  title: {
    fontSize: 40,
    fontWeight: 'bold',
    marginBottom: 10,
  },
  subtitle: {
    fontSize: 16,
    textAlign: 'center',
    marginBottom: 40,
  },
  imageSpace: {
    width: '80%',
    height: 180,
    backgroundColor: '#eee',
    borderRadius: 12,
    justifyContent: 'center',
    alignItems: 'center',
  },
  image: {
    width: '100%',
    height: '100%',
    resizeMode: 'contain'
  }
})
```

```
});
```

style.tsx

```
import { color } from "@hisius/ui/theme/colors";
import { StyleSheet, View } from "react-native";

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: color.background,
    alignItems: "center",
    justifyContent: "center",
  },
});

export { styles };
```

riskColor.ts

```
import { color } from "@hisius/ui/theme/colors";
import { ManchesterTriage } from "@hisius/enums";

export const getRiskColor = (risk: ManchesterTriage | null) => {
  if (risk === null) {
    return color.triage.nonUrgent;
  }

  switch (risk) {
    case ManchesterTriage.Emergency:
      return color.triage.emergency;

    case ManchesterTriage.VeryUrgent:
      return color.triage.veryUrgent;

    case ManchesterTriage.Urgent:
      return color.triage.urgent;

    case ManchesterTriage.Standard:
      return color.triage.standard;

    case ManchesterTriage.NonUrgent:
      return color.triage.nonUrgent;
  }
}
```

```

default:
    return color.triage.nonUrgent;
}
};

```

scale.ts

```

import { Dimensions } from "react-native";

const { width } = Dimensions.get("window");
const guidelineBaseWidth = 360;

export function scale(size: number) {
    return (width / guidelineBaseWidth) * size;
}

export function moderateScale(size: number, factor = 0.5) {
    return size + (scale(size) - size) * factor;
}

```

validate.ts

```

export const validateEmail = (email: string): boolean => {
    const regex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    return regex.test(email);
}

```

PACKAGES

navigation.ts

```

export type AuthStackParamList = {
    Splash: undefined;
    Signin: undefined;
    Login: undefined;
};

```

```

export type AppStackParamList = {
    Home: {
        id: string;
    };
    Screening: {
        id: string
    }
}

```

```

};

Atendance: {
  id: string
};

Profile: {
  id: string,
  name: string,
  email: string,
  password: string
};

}
}

```

//quando for navegar para alguma dessas telas elas precisam receber os
//parametros tipados aqui↓

user.d.ts

```

export type User = {
  id: number;
  name?: string;
  email: string;
  password: string;
  role: number;
  deleted: boolean;

  data_criacao: Date;
  data_atualizacao: Date;
}

```

userLogin.ts

```

export type userLogin = {
  email: string,
  password: string
}

```

userRegister.ts

```

export type userRegister = {
  name: string,
  email: string,
  password: string,
  confirmPassword: string,
  token?: string
}

```

axiosInstance.ts

```
// src/api/axiosInstance.ts
import axios from "axios";

const api = axios.create({
  baseURL: "http://localhost:8088",
  withCredentials: true,
});

export default api;
```

index.ts

```
export enum ManchesterTriage {
  Emergency = "imediato",
  VeryUrgent = "muito urgente",
  Urgent = "urgente",
  Standard = "pouco urgente",
  NonUrgent = "não urgente",
}
```

index.ts

```
import { ManchesterTriage } from "packages/enums/src";

export interface PatientCalledData {
  message: string;
  room: string;
}

export interface SocketEvents {
  "paciente-chamado": (data: PatientCalledData) => void;
  connect: () => void;
  disconnect: () => void;
  connect_error: (error: Error) => void;
  [key: string]: (...args: any[]) => void;
}

export interface UseSocketReturn {
  isConnected: boolean;
  lastNotification: PatientCalledData | null;
  emitEvent: <T>(event: string, data: T) => void;
}
```

```
export interface IPatient {  
    id?: number;  
    gender: "MASCULINO" | "FEMININO";  
    name: string;  
    cpf: string;  
    birthDate: string;  
    cnsNumber: string;  
    email: string;  
    phone: string;  
    motherName: string;  
    dateHourAttendance?: string;  
    age?: number;  
    attendanceId?: number;  
    classification?: ManchesterTriage;  
    position?: number;  
}
```

```
export interface IQueuedInfo {  
    id: number;  
    classification: ManchesterTriage;  
    estimatedWaitMinutes: number;  
    roomCalled: string;  
    queueType: string;  
}
```

```
export interface ApiResponse {  
    success: boolean;  
    message: string;  
    statusCode: number;  
    data?: any;  
    errors?: ApiError[];  
}
```

```
export interface UserResponse {  
    users: User[];  
    pagination: Pagination;  
}
```

```
export interface ApiError {  
    field: string;  
    message: string;  
}
```

```
export interface AvgTime {
```

```
averageWaitTime: number;
count: number;
}

export interface PeakDemand {
DOM?: number;
SEG?: number;
TER?: number;
QUA?: number;
QUI?: number;
SEX?: number;
SAB?: number;
}

export interface LogData {
id: number;
userId: number;
action: string;
module: string;
originIp: string;
userAgent: string;
createdAt: string;
}

export interface IEmployee {
name: string;
}

export interface ReportInfo {
avgTimeTreatmentInSec: AvgTime[];
avgTimeTriageInSec: AvgTime[];
peakDemand: PeakDemand;
}

export interface User {
id: number;
name: string;
email: string;
role: number;
}

export interface Pagination {
currentPage: number;
totalPages: number;
totalItems: number;
}
```

```

itemsPerPage: number;
hasNext: boolean;
hasPrev: boolean;
nextPage: number | null;
prevPage: number | null;
}

```

appNavigator.tsx

```

import { createNativeStackNavigator } from "@react-navigation/native-stack";
import { AppStackParamList } from "../../packages/@types/navigation";
import example from "apps/mobile/src/screens/example";

const Stack = createNativeStackNavigator<AppStackParamList>();

function AppNavigator() {
  return (
    <Stack.Navigator initialRouteName="Home" id={undefined}>
      <Stack.Screen name="Home" component={example} />
      {/* 
        <Stack.Screen name="Profile" component={Profile} />
        <Stack.Screen name="Screening" component={Screening} />
        <Stack.Screen name="Atendance" component={Atendance} />
      */}
      </Stack.Navigator>
  );
}

export default AppNavigator;

```

authNavigator.tsx

```

import { createNativeStackNavigator } from "@react-navigation/native-stack";
import Login from "apps/mobile/src/screens/login";
import { AuthStackParamList } from "../../packages/@types/navigation";

const Stack = createNativeStackNavigator<AuthStackParamList>();

function AuthNavigator() {
  return (
    <Stack.Navigator
      initialRouteName="Login"
      screenOptions={{ headerShown: false }}
      id={undefined}
    >
  
```

```

    >
    <Stack.Screen name="Login" component={Login} />
    {/* <Stack.Screen name="Register" component={Register} /> */}
    </Stack.Navigator>
);
}

```

```
export default AuthNavigator;
```

Admin.ts

```

import api from "./config/axios";
import type {
  UserResponse,
  ReportInfo,
  LogData,
  Pagination,
  ApiResponse,
  User,
} from "@hisius/interfaces";

```

```

interface LogsResponse {
  logs: LogData[];
  pagination: Pagination;
}

```

```

interface LogsResponse {
  users: User[];
  pagination: Pagination;
}

```

```

interface HospitalInfo {
  hospitalCode: string;
}

```

```

export class Admin {
  async getHospitalInfo(): Promise<HospitalInfo> {
    const response = await api.get<ApiResponse>(`admins/hospital-info`);
    return response.data.data;
  }
}

```

```

async changeUserRole(id: number, newRole: number): Promise<ApiResponse> {
  const response = await api.put<ApiResponse>(`users/${id}`, {
    role: newRole,
  });
}

```

```
    return response.data;
}

async getReport(startDate: string, endDate: string): Promise<ReportInfo> {
  const response = await api.get<ApiResponse>(`reports/`, {
    params: {
      startDate,
      endDate,
    },
  });
  return response.data.data;
}

async getEmployees(
  nameFilter?: string,
  page: number = 0,
  limit: number = 10
): Promise<UserResponse> {
  const params: Record<string, string> = {};
  if (nameFilter && nameFilter.trim() !== "") {
    params.nameFilter = nameFilter.trim();
  }
  params.page = page.toString();
  params.limit = limit.toString();
  const response = await api.get<ApiResponse>(`/employees`, { params });
  return response.data.data;
}

async getEmployeeRegisterCode(): Promise<string> {
  const response = await api.post<ApiResponse>(`/admins/staff-code`);

  return response.data.data.token;
}

async getLogs(
  page?: number,
  limit?: number,
  userId?: number,
  action?: string,
  module?: string
```

```

): Promise<LogsResponse> {
  const config = {
    params: {
      ...(page && { page }),
      ...(limit && { limit }),
      ...(userId && { userId }),
      ...(action && { action }),
      ...(module && { module }),
    },
  };
}

const response = await api.get<ApiResponse>(`/logs`, config);
return response.data.data;
}

async getAdmins(
  nameFilter?: string,
  page: number = 0,
  limit: number = 10
): Promise<UserResponse> {
  const params: Record<string, string> = {};
  if (nameFilter && nameFilter.trim() !== "") {
    params.nameFilter = nameFilter.trim();
  }
  params.page = page.toString();
  params.limit = limit.toString();

  const response = await api.get<ApiResponse>(`/admins`, { params });

  return response.data.data;
}
}

```

Auth.ts

```

import api from "./config/axios";
import type { userLogin } from "../../@types/userLogin";
import type { userRegister } from "../../@types/userRegister";
import { ApiResponse, User } from "@hisius/interfaces/src";

interface LoginResponse {
  id: number;
  name: string;

```

```
email: string;
role: number;
accessToken: string | null;
}

export class Auth {
  getAuthHeaders = (token: string) => ({
    headers: {
      Authorization: `Bearer ${token}`,
    },
  });
}

async changeName(name: string): Promise<LoginResponse> {
  const response = await api.put<ApiResponse>('/users/me', { name });
  return response.data.data;
}

async Login(userData: userLogin): Promise<LoginResponse> {
  const response = await api.post<ApiResponse>('/auth/login', userData);
  return response.data.data;
}

async getProfile(): Promise<User> {
  const response = await api.get<ApiResponse>('/users/me');
  return response.data.data;
}

async employeeRegister(
  userData: userRegister,
  token: string
): Promise<LoginResponse> {
  const response = await api.post<ApiResponse>(
    '/employees/',
    userData,
    this.getAuthHeaders(token)
  );
  return response.data.data;
}

async adminRegister(userData: userRegister): Promise<LoginResponse> {
  const response = await api.post<ApiResponse>('/admins', userData);
  return response.data.data;
}
```

```

async register(userData: userRegister): Promise<LoginResponse> {
  const response = await api.post<ApiResponse>('/users', userData);
  return response.data.data;
}

async resetEmail(token: string) {
  const response = await api.put<ApiResponse>(
    `/auth/confirm-change-email`,
    {},
    this.getAuthHeaders(token)
  );
  return response.data;
}

async resetPass(token: string, password: string, confirmPassword: string) {
  const response = await api.put<ApiResponse>(
    `/auth/recover-password`,
    { password, confirmPassword },
    this.getAuthHeaders(token)
  );
  return response.data;
}

async changeEmail(newEmail: string) {
  const response = await api.post<ApiResponse>(`/auth/change-email-request`, {
    email: newEmail,
  });
  return response.data;
}

async changePass(email: string) {
  const response = await api.post<ApiResponse>(`/auth/forgot-password`, {
    email,
  });
  return response.data;
}

```

index.ts

```

export * from "./Admin";
export * from "./Auth";
export * from "./Patient";
export * from "./Queue";

```

```
export * from "./helpers/asyncStorage";
export * from "./hooks/useSocket";
```

Patient.ts

```
import api from "./config/axios";
import type { ApiResponse, IPatient, IQueuedInfo } from "@hisius/interfaces";

export class Patient {
  async getQueueInfo(): Promise<IQueuedInfo> {
    const response = await api.get<ApiResponse>(`queue/me`);
    return response.data.data;
  }

  async getProfile(): Promise<IPatient> {
    const response = await api.get<ApiResponse>(`patients/me`);
    return response.data.data;
  }

  async createProfile(patient: IPatient): Promise<boolean> {
    const response = await api.post<ApiResponse>(`patients`, patient);
    return response.data != null;
  }

  async updateProfile(patient: IPatient): Promise<boolean> {
    const response = await api.put<ApiResponse>(`patients/me`, patient);
    return response.data != null;
  }

  async leaveQueue() {
    const response = await api.delete<ApiResponse>(`queue/leave`);
    return response.data != null;
  }
}
```

Queue.ts

```
import api from "./config/axios";
import type {
  ApiError,
  ApiResponse,
  IPatient,
  Pagination,
```

```
        } from "@hisius/interfaces";

interface QueueResponse {
    patients: IPatient[];
    pagination: Pagination;
}

export class Queue {
    async getPatientInRoomByQueue(
        isTriage: boolean,
        nameFilter?: string
    ): Promise<QueueResponse> {
        const params: Record<string, string> = {};

        if (nameFilter && nameFilter.trim() !== "") {
            params.nameFilter = nameFilter.trim();
        }

        const response = await api.get<ApiResponse>(
            `/queue/${isTriage ? "triage" : "treatment"}/room`,
            Object.keys(params).length > 0 ? { params } : {}
        );

        return response.data.data;
    }

    async getPatientByQueue(
        isTriage: boolean,
        nameFilter?: string
    ): Promise<QueueResponse> {
        const params: Record<string, string> = {};

        if (nameFilter && nameFilter.trim() !== "") {
            params.nameFilter = nameFilter.trim();
        }

        const response = await api.get<ApiResponse>(
            `/queue/${isTriage ? "triage" : "treatment"}/patients`,
            Object.keys(params).length > 0 ? { params } : {}
        );

        return response.data.data;
    }
}
```

```

async joinQueue(hospitalCode: string): Promise<boolean> {
  const response = await api.post<ApiResponse>(`/queue/join`, {
    hospitalCode,
  });
  return response.status >= 200 && response.status < 300;
}

async getPatient(id: number): Promise<IPatient> {
  const response = await api.get<ApiResponse>(`/patients/${id}`);
  return response.data.data;
}

async finishTreatment(userId: number): Promise<ApiResponse> {
  const response = await api.delete<ApiResponse>(`/queue/${userId}/finish`);
  return response.data.data;
}

async getNextPatient(isTriage: boolean, room: string): Promise<IPatient> {
  const response = await api.post<ApiResponse>(
    `/queue/${isTriage ? "triage" : "treatment"}/call-next`,
    { room }
  );
  return response.data.data;
}

async updateClassification(
  id: number,
  classification: string
): Promise<IPatient | ApiError[]> {
  const response = await api.put<ApiResponse>(`/queue/${id}`, {
    classification: classification.toLowerCase(),
  });

  return response.data.data;
}

async getQueueCount(type: string): Promise<number> {
  const response = await api.get<ApiResponse>(`/queue/${type}/count`);
  return response.data.data.count;
}

async goToTreatment(
  id: number,
  classification: string
)

```

```

}): Promise<IPatient | ApiError[]> {
  const response = await api.put<ApiResponse>(`/queue/${id}/next`, {
    classification: classification.toLowerCase(),
  });

  return response.data.data;
}
}
}

```

axios.ts

```

import axios from "axios";
import LocalStorageManager from "../helpers/localStorageManager";
import { refreshTokenInterceptor } from "./refreshInterceptors";
import { env } from "./env";
import { getToken } from "../helpers/asyncStorage";

const api = axios.create({
  baseURL: env.API_BASE_URL,
  timeout: env.API_TIMEOUT,
  headers: {
    "Content-Type": "application/json",
  },
  withCredentials: true,
});

const excludedRoutes = [
  "/auth/recover-password",
  "/auth/login",
  "/users",
  "/employees/",
  "/auth/confirm-change-email",
];

```

api.interceptors.request.use(
 async (config) => {
 const isExcludedRoute = excludedRoutes.some(
 (route) => config.url === route
);

```

    const token = LocalStorageManager.getAccessToken() || (await getToken());
    if (token && !isExcludedRoute) {
      config.headers.Authorization = `Bearer ${token}`;
    }
  }
}
```

```

if (process.env.NODE_ENV === "development") {
  console.log(
    ` ${
      config.method?.toUpperCase() } ${config.url}`,
    config.data || ""
  );
}

return config;
},
(error) => {
  return Promise.reject(error);
}
);

api.interceptors.response.use((response) => {
  if (process.env.NODE_ENV === "development") {
    console.log(
      ` ${
        response.config.method?.toUpperCase() } ${response.config.url}`,
      response.data
    );
  }
  return response;
}, refreshTokenInterceptor);

export default api;

```

env.ts

```

export const env = {
  API_BASE_URL: "http://localhost:8088",
  API_TIMEOUT: 10000,
  IS_DEV: "development",
};

```

refreshInterceptors.ts

```

import axios from "axios";
import LocalStorageManager from "../helpers/localStorageManager";
import { logout, saveToken } from "../helpers/asyncStorage";
import { env } from "./env";

let isRefreshing = false;

```

```

let failedRequestsQueue = [];

const processQueue = (error, token = null) => {
  failedRequestsQueue.forEach((request) => {
    if (error) {
      request.reject(error);
    } else {
      request.resolve(token);
    }
  });
  failedRequestsQueue = [];
};

export const refreshTokenInterceptor = async (error) => {
  const originalRequest = error.config;

  const excludedRoutes = [
    "/auth/recover-password",
    "/auth/login",
    "/users",
    "/employees/",
    "/auth/confirm-change-email",
  ];
}

const shouldSkipRefresh = excludedRoutes.some(
  (route) => originalRequest.url === route
);

if (shouldSkipRefresh) {
  return Promise.reject(error);
}

if (!originalRequest._retry) {
  if (error.response?.status === 401 || error.response?.status === 403) {
    if (isRefreshing) {
      return new Promise((resolve, reject) => {
        failedRequestsQueue.push({ resolve, reject });
      })
        .then((token) => {
          originalRequest.headers.Authorization = `Bearer ${token}`;
          return axios(originalRequest);
        })
        .catch((err) => Promise.reject(err));
    }
  }
}

```

```
originalRequest._retry = true;
isRefreshing = true;

try {
  const response = await axios.post(
    `${env.API_BASE_URL}/auth/refresh`,
    null,
    { withCredentials: true }
  );

  const accessToken = response.data.data.accessToken;

  LocalStorageManager.setTokens(accessToken);
  await saveToken(accessToken);
  axios.defaults.headers.Authorization = `Bearer ${accessToken}`;

  processQueue(null, accessToken);

  originalRequest.headers.Authorization = `Bearer ${accessToken}`;
  return axios(originalRequest);
} catch (refreshError) {
  processQueue(refreshError, null);

  LocalStorageManager.clearAuth();
  await logout();
  delete axios.defaults.headers.Authorization;

  if (window.location.pathname !== "/login") {
    window.location.href = "/login";
  }

  return Promise.reject(refreshError);
} finally {
  isRefreshing = false;
}

return Promise.reject(error);
};
```

asyncStorage.ts

```
import AsyncStorage from "@react-native-async-storage/async-storage";
import { User } from "@hisius/interfaces/src";
import LocalStorageManager from "./localStorageManager";

export const saveToken = async (token: string) => {
  await AsyncStorage.setItem("token", token);
};

export const getToken = async () => {
  return await AsyncStorage.getItem("token");
};

export const getUser = async () => {
  return await AsyncStorage.getItem("user");
};

export const saveUser = async (user: User) => {
  await AsyncStorage.setItem("user", JSON.stringify(user));
};

export const logout = async () => {
  await AsyncStorage.removeItem("token");
  await AsyncStorage.removeItem("user");
  await LocalStorageManager.clearAll();
};
```

localStorageManager.ts

```
const STORAGE_KEYS = {
  ACCESS_TOKEN: "accessToken",
  USER: "user",
};

class LocalStorageManager {
  static setTokens(accessToken) {
    this.setItem(STORAGE_KEYS.ACCESS_TOKEN, accessToken);
  }

  static getAccessToken() {
    return this.getItem(STORAGE_KEYS.ACCESS_TOKEN);
  }
}
```

```
static clearTokens() {
    this.removeItem(STORAGE_KEYS.ACCESS_TOKEN);
}

static setUser(user) {
    this.setItem(STORAGE_KEYS.USER, JSON.stringify(user));
}

static getUser() {
    const user = this.getItem(STORAGE_KEYS.USER);
    return user ? JSON.parse(user) : null;
}

static clearUser() {
    this.removeItem(STORAGE_KEYS.USER);
}

static setItem(key, value) {
    try {
        if (typeof window !== "undefined") {
            localStorage.setItem(key, value);
        }
    } catch (error) {
        console.error(`Erro ao salvar no localStorage [${key}]:`, error);
    }
}

static getItem(key) {
    try {
        if (typeof window !== "undefined") {
            return localStorage.getItem(key);
        }
    } catch (error) {
        console.error(`Erro ao ler do localStorage [${key}]:`, error);
        return null;
    }
}

static removeItem(key) {
    try {
        if (typeof window !== "undefined") {
            localStorage.removeItem(key);
        }
    }
```

```
        } catch (error) {
            console.error(`Erro ao remover do localStorage [${key}]:`, error);
        }
    }

static clear() {
    try {
        if (typeof window !== "undefined") {
            localStorage.clear();
        }
    } catch (error) {
        console.error("Erro ao limpar localStorage:", error);
    }
}

static hasToken() {
    return !!this.getAccessToken();
}

static isAuthenticated() {
    return this.hasToken();
}

static clearAuth() {
    this.clearTokens();
    this.clearUser();
}

static clearAll() {
    this.clear();
}

static getAllItems() {
    try {
        if (typeof window !== "undefined") {
            const items = {};
            for (let i = 0; i < localStorage.length; i++) {
                const key = localStorage.key(i);
                if (key) {
                    items[key] = localStorage.getItem(key);
                }
            }
            return items;
        }
    }
}
```

```

        return {};
    } catch (error) {
        console.error("Erro ao obter todos os itens do localStorage:", error);
        return {};
    }
}

static getSize() {
    try {
        if (typeof window !== "undefined") {
            let total = 0;
            for (let key in localStorage) {
                if (localStorage.hasOwnProperty(key)) {
                    total += localStorage[key].length + key.length;
                }
            }
            return total;
        }
        return 0;
    } catch (error) {
        console.error("Erro ao calcular tamanho do localStorage:", error);
        return 0;
    }
}
}

export default LocalStorageManager;
export { STORAGE_KEYS };

```

useSocket.ts

```

import { useEffect, useRef, useState } from "react";
import io, { Socket } from "socket.io-client";

interface PatientCalledData {
    message: string;
    room: string;
}

interface UseSocketReturn {
    isConnected: boolean;
    lastNotification: PatientCalledData | null;
    emitEvent: <T>(event: string, data: T) => void;
}

```

```

const SOCKET_SERVER_URL = "http://localhost:8088";

export const useSocket = (
  userId: number | null,
  authToken: string | null
): UseSocketReturn => {
  const socketRef = useRef<any | null>(null);
  const [isConnected, setIsConnected] = useState<boolean>(false);
  const [lastNotification, setLastNotification] =
    useState<PatientCalledData | null>(null);

  useEffect(() => {
    if (!userId || !authToken) {
      console.log("Socket: Aguardando userId ou authToken...");
      if (socketRef.current) {
        socketRef.current.disconnect();
        socketRef.current = null;
        setIsConnected(false);
      }
      return;
    }
    console.log("Iniciando conexão WebSocket para usuário:", userId);

    const bearerToken = authToken.startsWith("Bearer ")
      ? authToken
      : `Bearer ${authToken}`;

    socketRef.current = io(SOCKET_SERVER_URL, {
      transports: ["websocket", "polling"],
      auth: {
        authorization: bearerToken,
      },
      query: {
        userId: userId.toString(),
      },
      transportOptions: {
        polling: {
          extraHeaders: {
            Authorization: bearerToken,
          },
        },
      },
    });
  });
}

```

```
});

const socket = socketRef.current;

socket.on("connect", () => {
  console.log("Conectado ao WebSocket");
  setIsConnected(true);
  socket.emit("join-user-room", `user:${userId}`);
});

socket.on("disconnect", (reason: string) => {
  console.log("Desconectado do WebSocket. Razão:", reason);
  setIsConnected(false);
});

socket.on("connect_error", (error: Error) => {
  console.error("Erro de conexão:", error.message);
  setIsConnected(false);
});

socket.on("authenticated", () => {
  console.log("Socket autenticado com sucesso");
});

socket.on("unauthorized", (error: any) => {
  console.error("Falha na autenticação:", error);
  setIsConnected(false);
});

socket.on("paciente-chamado", (data: PatientCalledData) => {
  console.log("Paciente chamado via WebSocket:", data);
  setLastNotification(data);
});

return () => {
  if (socketRef.current) {
    console.log("Limpando conexão WebSocket");
    socketRef.current.disconnect();
    socketRef.current = null;
    setIsConnected(false);
  }
};
}, [userId, authToken]);
```

```

const emitEvent = <T>(event: string, data: T): void => {
  if (socketRef.current && isConnected) {
    socketRef.current.emit(event, data);
  } else {
    console.warn("Tentativa de emitir evento sem conexão ativa");
  }
};

return {
  isConnected,
  lastNotification,
  emitEvent,
};
};

```

index.ts

```

export * from "./components/Button";
export * from "./components/CustomInput";

```

index.tsx

```

import { useState } from "react";
import { TouchableOpacity, Text, LayoutAnimation } from "react-native";
import { styles } from "./styles";

const CustomButton = ({ title, onPress, style = {}, disabled = false }) => {
  const [isPressed, setIsPressed] = useState(false);

  const handlePressIn = () => {
    if (!disabled) {
      setIsPressed(true);
      LayoutAnimation.configureNext(
        LayoutAnimation.create(
          100,
          LayoutAnimation.Types.easeInEaseOut,
          LayoutAnimation.Properties.scaleXY
        )
      );
    }
  };

  const handlePressOut = () => {

```

```

if (!disabled) {
  setIsPressed(false);
  LayoutAnimation.configureNext(
    LayoutAnimation.create(
      100,
      LayoutAnimation.Types.easeInEaseOut,
      LayoutAnimation.Properties.scaleXY
    )
  );
}

};

const getButtonStyle = () => {
  if (disabled) return [styles.button, styles.buttonDisabled];

  const styleArray = [styles.button];
  if (isPressed) {
    styleArray.push({
      transform: [{ scale: 0.99 }],
    });
  }

  return styleArray;
};

const getTextStyle = () => {
  return disabled ? [styles.text, styles.textDisabled] : [styles.text];
};

return (
  <TouchableOpacity
    style={[...getButtonStyle(), style]}
    onPress={onPress}
    disabled={disabled}
    onPressIn={handlePressIn}
    onPressOut={handlePressOut}
    activeOpacity={1}
  >
  <Text style={getTextStyle()}>{title}</Text>
  </TouchableOpacity>
);
};

export default CustomButton;

```

styles.tsx

```
import { color } from "@hisius/ui/theme/colors";
import { StyleSheet, ViewStyle, TextStyle } from "react-native";

interface ButtonStyles {
  button: ViewStyle;
  buttonHover: ViewStyle;
  buttonPressed: ViewStyle;
  buttonDisabled: ViewStyle;
  text: TextStyle;
  textDisabled: TextStyle;
}

const baseButton: ViewStyle = {
  backgroundColor: color.primary,
  paddingVertical: 14,
  paddingHorizontal: 32,
  borderRadius: 6,
  height: 50,
  alignItems: "center",
  justifyContent: "center",
};

const styles = StyleSheet.create<ButtonStyles>({
  button: baseButton,

  buttonHover: {
    ...baseButton,
  },

  buttonPressed: {
    ...baseButton,
    transform: [{ scale: 0.99 }],
  },

  buttonDisabled: {
    ...baseButton,
    backgroundColor: color.deactive,
    borderColor: color.gray,
  },
  text: {
```

```

color: color.front,
fontSize: 15,
fontFamily: "Montserrat",
fontWeight: "400",
letterSpacing: 0.5,
},

textDisabled: {
  color: color.gray,
  fontSize: 16,
  fontFamily: "Montserrat",
  fontWeight: "600",
  letterSpacing: 0.5,
},
});

export { styles };

```

index.tsx

```

import React, { useState, useRef } from "react";
import {
  View,
  TextInput,
  Text,
  KeyboardTypeOptions,
  StyleProp,
  TextStyle,
  ViewStyle,
  Platform,
  TouchableOpacity,
} from "react-native";

import { styles } from "./styles";

interface CustomInputProps {
  value: string;
  style?: StyleProp<ViewStyle>;
  inputStyle?: StyleProp<TextStyle>;
  onChangeText: (text: string) => void;
  placeholder?: string;
  secureTextEntry?: boolean;
  keyboardType?: KeyboardTypeOptions;
  autoCapitalize?: "none" | "sentences" | "words" | "characters";
}

```

```

error?: string;
icon?: React.ReactNode;
visibilityOff?: React.ReactNode;
visibilityOn?: React.ReactNode;
onIconPress?: () => void;
disabled?: boolean;
inputType?: string;
inputId?: string;
maxLength?: number;
onSubmitEditing?: () => void;
returnKeyType?: "done" | "go" | "next" | "search" | "send";
}

const CustomInput: React.FC<CustomInputProps> = ({
  value,
  style,
  inputStyle,
  onChangeText,
  placeholder,
  secureTextEntry = false,
  keyboardType = "default",
  autoCapitalize = "sentences",
  error = "",
  icon,
  onIconPress,
  disabled = false,
  inputType,
  inputId,
  onSubmitEditing,
  maxLength,
  returnKeyType = "done",
  visibilityOff,
  visibilityOn,
  ...props
}) => {
  const [isFocused, setIsFocused] = useState(false);
  const inputRef = useRef<TextInput>(null);
  const [isPasswordVisible, setIsPasswordVisible] = useState(false);

  const showLabel = isFocused || value !== "";
  
```

const handleWebStyling = () => {
 if (Platform.OS === "web" && inputRef.current) {
 const element = inputRef.current as any;

```
if (element.setNativeProps) {
  element.setNativeProps({
    style: {
      border: "none",
      outline: "none",
      boxShadow: "none",
    },
  });
}

const togglePasswordVisibility = () => {
  setIsPasswordVisible(!isPasswordVisible);
};

const renderPasswordIcon = () => {
  return (
    <TouchableOpacity
      onPress={togglePasswordVisibility}
      style={styles.iconContainer}
      disabled={disabled}
    >
      {isPasswordVisible ? visibilityOn : visibilityOff}
    </TouchableOpacity>
  );
};

return (
  <View style={[styles.container, style]}>
    <View
      style={[
        styles.inputContainer,
        error ? styles.inputContainerError : null,
        isFocused ? styles.inputContainerFocused : null,
        disabled ? styles.inputContainerDisabled : null,
      ]}
    >
      {icon && <View style={styles.iconContainer}>{icon}</View>}

      <View style={styles.inputWrapper}>
        {showLabel && placeholder && (
          <Text
            style={[
              styles.floatingLabel,
            ]
          >
            {placeholder}
          </Text>
        )}
      </View>
    
```

```
        disabled ? styles.floatingLabelDisabled : null,
    ]}
>
  {placeholder}
</Text>
)}
```



```
<TextInput
  ref={inputRef}
  style={[
    styles.input,
    inputStyle,
    showLabel ? styles.inputWithLabel : null,
    disabled ? styles.inputDisabled : null,
    Platform.OS === "web" && {
      borderWidth: 0,
      outlineWidth: 0,
    },
  ]}
  value={value}
  onChangeText={onChangeText}
  placeholder={!showLabel ? placeholder : ""}
  secureTextEntry={secureTextEntry && !isPasswordVisible}
  placeholderTextColor="#999"
  selectionColor="#007AFF"
  keyboardType={keyboardType}
  autoCapitalize={autoCapitalize}
  onSubmitEditing={onSubmitEditing}
  returnKeyType={returnKeyType}
  maxLength={maxLength}
  onFocus={() => {
    if (!disabled) {
      setIsFocused(true);
      handleWebStyling();
    }
  }}
  onBlur={() => setIsFocused(false)}
  underlineColorAndroid="transparent"
  textAlignVertical="center"
  onLayout={handleWebStyling}
  editable={!disabled}
  selectTextOnFocus={!disabled}
  {...(inputType && { "data-type": inputType })}
  {...(inputId && { "data-id": inputId })}
```

```

        {...(inputId && { testID: inputId })}
        {...props}
    />
</View>
{secureTextEntry && renderPasswordIcon()}
</View>

{error ? <Text style={styles.errorText}>{error}</Text> : null}
</View>
);
};

export default CustomInput;

```

styles.tsx

```

import { StyleSheet, Platform } from "react-native";

export const styles = StyleSheet.create({
  container: {
    marginBottom: 0,
  },
  label: {
    fontSize: 16,
    fontWeight: "600",
    color: "#333",
    marginBottom: 8,
  },
  inputContainer: {
    borderWidth: 1,
    borderColor: "#ddd",
    borderRadius: 8,
    backgroundColor: "#fff",
    height: 56,
    flexDirection: "row",
    alignItems: "flex-start",
    shadowColor: "#000",
    shadowOffset: {
      width: 0,
      height: 1,
    },
    shadowOpacity: 0.1,
    shadowRadius: 2,
    flex: 1,
  }
});

```

```
minWidth: "100%",  
overflow: "hidden",  
},  
inputContainerError: {  
  borderColor: "#FF3B30",  
},  
inputContainerFocused: {  
  borderColor: "#007AFF",  
  borderWidth: 2,  
},  
inputWrapper: {  
  flex: 1,  
  position: "relative",  
  justifyContent: "center",  
},  
floatingLabel: {  
  position: "absolute",  
  top: 6,  
  left: 12,  
  fontSize: 12,  
  fontWeight: "500",  
  color: "#007AFF",  
  backgroundColor: "#fff",  
  paddingHorizontal: 4,  
  zIndex: 1,  
},  
input: {  
  fontSize: 16,  
  color: "#333",  
  padding: 16,  
  height: "100%",  
  flex: 1,  
  minWidth: 0,  
  includeFontPadding: false,  
  textAlignVertical: "center",  
  backgroundColor: "transparent",  
  borderWidth: 0,  
  borderColor: "transparent",  
},  
inputWithLabel: {  
  paddingTop: 20,  
  paddingBottom: 12,  
},  
inputContainerDisabled: {
```

```

backgroundColor: "#f5f5f5",
borderColor: "#e0e0e0",
},
inputDisabled: {
  color: "#999",
},
floatingLabelDisabled: {
  color: "#999",
},
iconContainer: {
  padding: 16,
  justifyContent: "center",
  alignItems: "center",
  width: 48,
  height: "100%",
},
errorText: {
  color: "#FF3B30",
  fontSize: 14,
  marginTop: 4,
  marginLeft: 4,
},
});

```

colors.ts

```

export const color = {
  primary: "#395499",
  secondary: "#90CBF3",
  background: "#F3F3F3",
  card: "#FFFFFF",
  front: "#FFFFFF",
  text: "#0D1329",
  gray: "#D5D6D9",
  deactive: "#CCCCCC",
  triage: {
    emergency: "#E53E3E",
    veryUrgent: "#DD6B20",
    urgent: "#D69E2E",
    standard: "#38A169",
    nonUrgent: "#3182CE",
  },
};

```

```
error: {  
    error: "#E53E3E",  
    warning: "#D69E2E",  
    ok: "#38A169",  
},  
};
```