1. **Introdução ao Documento**

Este documento apresenta o projeto de desenvovimento de um website com o objetivo de avaliar a satisfação dos pacientes nas consultas realizadas pela psicóloga fictícia Dra. Ana Maria. O site visa oferecer um espaço simples e intuitivo para que os pacientes possam expressar sua opinião sobre a qualidade das consultas e do atendimento prestado, contribuindo assim para o aprimoramento contínuo dos serviços.

O projeto segue uma abordagem objetiva, com foco nas funcionalidades de avaliação de satisfação, sem a necessidade de múltiplas páginas ou elementos complexos. Além disso, o design do site será desenvolvido levando em consideração a identidade visual da psicóloga, que utiliza tons de azul claro e verde pastel, transmitindo uma sensação de tranquilidade e acolhimento.

* 1. **Tema**

O tema deste projeto é a criação de uma plataforma digital para mensurar a satisfação dos pacientes em relação às consultas psicológicas realizadas pela Dra. Ana Maria. A plataforma se concentrará exclusivamente em permitir que os pacientes façam uma avaliação rápida e objetiva sobre a qualidade das consultas, promovendo uma interação prática e eficiente.

* 1. **Objetivo do Projeto**

O objetivo geral deste projeto é desenvolver uma plataforma online que permita aos pacientes da Dra. Ana Maria avaliarem suas consultas, promovendo uma análise contínua da qualidade do atendimento psicológico. O sistema visa ser uma ferramenta simples e eficaz para medir a satisfação dos pacientes, possibilitando o aprimoramento dos serviços com base no feedback recebido.

**Objetivos Específicos (Requisitos Funcionais):**

* Facilitar a coleta de feedbacks: Oferecer uma interface intuitiva e acessível para que os pacientes possam avaliar suas consultas de maneira rápida e objetiva.
* Promover melhorias no atendimento: Utilizar os dados de satisfação para identificar oportunidades de melhorias nos serviços oferecidos, focando no bem-estar e satisfação do paciente.
* Preservar a identidade visual: Desenvolver o site com base na identidade visual da Dra. Ana Maria, utilizando cores que transmitem tranquilidade e acolhimento.

**Requisitos Não Funcionais:**

* Desempenho: O sistema deve ser capaz de processar até 100 avaliações por minuto sem impactar a performance.
* Segurança: Todos os dados de feedback devem ser armazenados de forma segura, seguindo os padrões de criptografia.
* Usabilidade: A interface do sistema deve ser intuitiva, com um design limpo e minimalista, para que usuários de todas as idades possam utilizá-la sem dificuldades.
* Compatibilidade: O sistema deve ser compátivel com diferentes dispositivos e navegadores (responsividade), funcionando em desktops, tablets e smartphones.
* Disponibilidade: O sistema deve estar disponível 99,9% do tempo, garantindo que os pacientes possam acessá-lo a qualquer momento para realizar suas avaliações.
* Manutenção: O sistema deve ser desenvolvido de forma modular, facilitando futuras atualizações e correções de bugs.
  1. **Delimitação do Problema**

O problema central deste projeto é a falta de uma ferramenta simples e eficaz para que a Dra. Ana Maria possa medir a satisfação de seus pacientes em relação às consultas psicológicas. Apesar de existirem métodos tradicionais para a coleta de feedback, como questionários impressos ou entrevistas presenciais, eles podem ser demorados e inviabilizar um acompanhamento contínuo e ágil. Portanto, o projeto busca desenvolver uma solução digital que automatize esse processo de avaliação, focando em uma interface fácil de usar e que ofereça dados confiáveis para aprimorar o atendimento da psicóloga.

* 1. **Justificativa da Escolha do Tema**

A escolha deste tema foi motivada pela crescente demanda por soluções tecnológicas no campo da psicologia, em especial no contexto da telemedicina e das consultas remotas, que se tornaram mais frequentes. O uso de plataformas digitais para coletar feedback de pacientes é uma prática cada vez mais adotada, pois permite um acompanhamento contínuo da qualidade do atendimento e facilita a implementação de melhorias. Este projeto, além de proporcionar uma ferramenta prática, contribui para o desenvolvimento de competências em criação de interfaces voltadas para o usuário e o tratamento de dados qualitativos de maneira segura.

* 1. **Método de Trabalho**

O projeto será desenvolvido utilizando uma metodologia ágil, com base em Scrum, permitindo iterações frequentes e entregas contínuas. A modelagem utilizada será orientada a objetos, com foco em modularidade e reutilização de componentes. Para o desenvolvimento do software, será adotada a arquitetura de aplicação frontend, com React no frontend para a criação da interface de usuário.

* 1. **Organização do Trabalho**

Este documento está organizado em capítulos que detalham o desenvolvimento do projeto. Na introdução, são apresentados o tema, objetivo delimitação do problema justificativa da escolha do tema, e método de trabalho. Os capítulos subsequentes abordam a análise de requisitos, a modelagem do sistema, o desenvolvimento técnico e conclusões finais, incluindo uma avaliação dos resultados obtidos. Também é incluído um glossário ao final para facilitar a compreensão de termos técnicos utilizados no projeto.

* 1. **Glossário**
* Frontend: Parte do sistema que interagem diretamente com o usuário (parte visual)
* Scrum: Metodologia ágil de gerenciamento de projetos, com ciclos de desenvolvimento curtos e entregas frequentes.
* React: Biblioteca JavaScript usada para a construção de interfaces de usuário.

1. **Descrição Geral do Sitema**
   1. **Descrição do Problema**

O problema central abordado pelo sistema é a ausência de um método digital prático para que a Dra. Ana Maria possa avaliar, de forma sistemática e eficaz, a satisfação de seus pacientes após as consultas. Atualmente, o feedback dos pacientes pode ser obtido por meios tradicionais, como conversas informais ou formulários físicos, que são menos eficientes e não permitam uma análise estruturada.

**Quem é afetado pelo sistema?**

O sistema afeta tanto a Dra. Ana Maria quanto seus pacientes, Dra. Ana Maria tendo uma média de como estão sendo seus atendimentos quanto seus pacientes conseguindo avaliar para que a mesma possa cada vez melhorar mais e atender as necessidades de seus pacientes.

**Qual é o impacto do sistema?**

O sistema terá um impacto significativo na qualidade do serviço presto, ao permitir que a Dra. Ana Maria compreenda melhor as percepções e expectativas de seus pacientes. Ao contar com uma ferramenta de avaliação digital, será possível coletar dados em tempo real e identificar pontos fortes e áreas que necessitam de melhorias. Além disso, o sistema promove maior engajamento dos pacientes, que se sentirão mais valorizados por terem suas opiniões levadas em consideração.

**Qual seria uma boa solução para o problema?**

A solução ideal é o desenvolvimento de um sistema online de feedback que permita aos pacientes avaliarem suas consultas de forma rápida e anônima. O sistema deve ser fácil de usar, acessível a partir de qualquer dispositivo (computadores, tablets, smartphones) e garantir que as informações coletadas sejam armazenadas de maneira segura e confidencial. A Dra. Ana Maria deve poder acessar essas informações por meio de um painel administrativo simples, que organize as avaliações e apresente relatórios claros sobre a satisfação geral dos pacientes.

* 1. **Principais Envolvidos e suas Características**
     1. **Usuários do Sistema**

Este software é destinado a clínicas de psicologia, psicólogos, psicanalistas e terapeutas. Além disso, atende os usuários finais, que são os pacientes desses profissionais.

* + 1. **Desenvolvedores do Sistema**

O Desenvolvimento do sistema é realizado por um único desenvolvedor, especializado em tecnologias web como React, Node.js e MongoDB. Este profissional é responsável por todas as etapas do desenvolvimento, desde a criação da interface até a implementação da lógica no servidor. Suas principais responsabilidades incluem:

* + **Desenvolvimento Frontend:** Criação da interface do usuário, garantindo que o sistema seja fácil de usar para psicólogos, terapeutas e pacientes.
  + **Desenvolvimento Backend:** Implementação da lógica do servidor, integração com o banco de dados e gerenciamento de segurança de dados.
  + **Manutenção e Suporte:** Realização de atualizações, correções de bugs e melhorias contínuas no sistema, sempre alinhadas às necessidades dos usuários.
  1. **Regras do Negócio**

1. **Restrições de Acesso:**

Apenas psicólogos e terapeutas cadastrados podem acessar as funcionalidades administrativas do sistema, como a visualização e o gerenciamento de informações dos pacientes. Pacientes tem acesso restrito apenas as páginas de avaliar a sessão.

1. **Armazenamento de Dados:**

Todos os dados de pacientes, informações pessoais e feedbacks, devem ser armazenados de maneira segura e criptografada. O sistema deve permitir o armazenamento de até 5GB de dados por clínica.

1. **Tolerância a Falhas:**

O sistema deve ser capaz de lidar com interrupções no serviço sem perda de dados.

1. **Restrições de Desempenho:**

O sistema foi testado com 4 requisições de feedback simultâneas e conseguiu processá-las sem qualquer atraso perceptível ou erro. Portanto, o sistema está otimizado para suportar, ao menos, um pequeno volume de interações simultâneas, como esperado para o uso típico de uma clínica de psicologia com número moderado de pacientes.

1. **Ferramentas de Apoio:**

O sistema utilizará ferramentas de backup automático semanal e monitoramento em tempo real para garantir a continuidade do serviço e a recuperação de dados em casa de falha.

1. **Requisitos do Sistema**
   1. Requisitos Funcionais
2. Feedback de Sessões

O sistema deve permitir os pacientes forneçam feedback sobre suas sessões, avaliando o atendimento de 1 a 5 estrelas.

1. Visualização de Feedbacks

O administrador (psicóloga) deve poder visualizar os feedbacks fornecidos pelos pacientes em uma interface amigável e organizada, preferencialmente em gráficos.

1. Relatórios e Gráficos

O sistema deve gerar relatórios e gráficos sobre o feedback dos pacientes, permitindo que a psicóloga acompanhe ao longo do tempo.

**Caso de Uso (Detalhamento dos requisitos funcionais)**

A diagram of a person's process

Description automatically generated

Caso de Uso 1: Envio de Feedback

* Ator Principal: Paciente
* Objetivo: Enviar feedback sobre uma sessão com a psicóloga.
* Fluxo Principal:
  + O paciente acessa o sistema após a sessão.
  + Navega até a página de feedback.
  + Avalia a sessão em uma escala de 1 a 5 estrelas.
  + Adiciona comentários opcionais.
  + Clicar em “Enviar”
  + O sistema armazena o feedback e exibe uma mensagem de confirmação.

Caso de Uso 2: Visualizar Feedbacks

* Ator Principal: Psicóloga
* Objetivo: Visualizar os feedbacks fornecidos pelos pacientes sobre as sessões.
* Pré-condição: A psicóloga estar autenticada no sistema e haver feedbacks registrados no banco de dados.
* Fluxo Principal:
  + A psicóloga faz login no sistema usando suas credenciais.
  + Navega até a página de “Dashboard”.
  + O sistema exibe uma lista de feedbacks recebidos, com informações como:
    - Nome do paciente (ou “Anônimo”, se o feedback for anônimo).
    - Avaliação de 1 a 5 estrelas.
    - Comentários adicionais (se houver).
    - Avaliação de 1 a 5 se voltaria para uma próxima sessão.
    - Data da sessão correspondente.
* O sistema exibe gráficos que mostram a média das avaliações ao longo do tempo e distribuições de notas, facilitando a análise de dados.
* Fluxo Alternativo:
  + Fluxo 1: Se não houver feedbacks disponíveis, o sistema exibe uma mensagem informando que não há feedbacks para serem visualizados no momento.
  + Fluxo 2: Se a psicóloga tenta acessar a página sem estar autenticada, o sistema redireciona para a página de login.
* Pós-condição: A psicóloga visualiza os feedbacks e gráficos de avaliação, obtendo uma visão geral das opiniões e da satisfação dos pacientes.
  1. Requisitos Não-Funcionais

1. Desempenho

O sistema deve ser capaz de processar múltiplas requisições simultâneas, garantindo que até 10 pacientes possam enviar feedbacks ao mesmo tempo sem queda de desempenho perceptível.

1. Segurança

O sistema deve garantir a segurança das informações dos pacientes, protegendo dados sensíveis.

1. Usabilidade

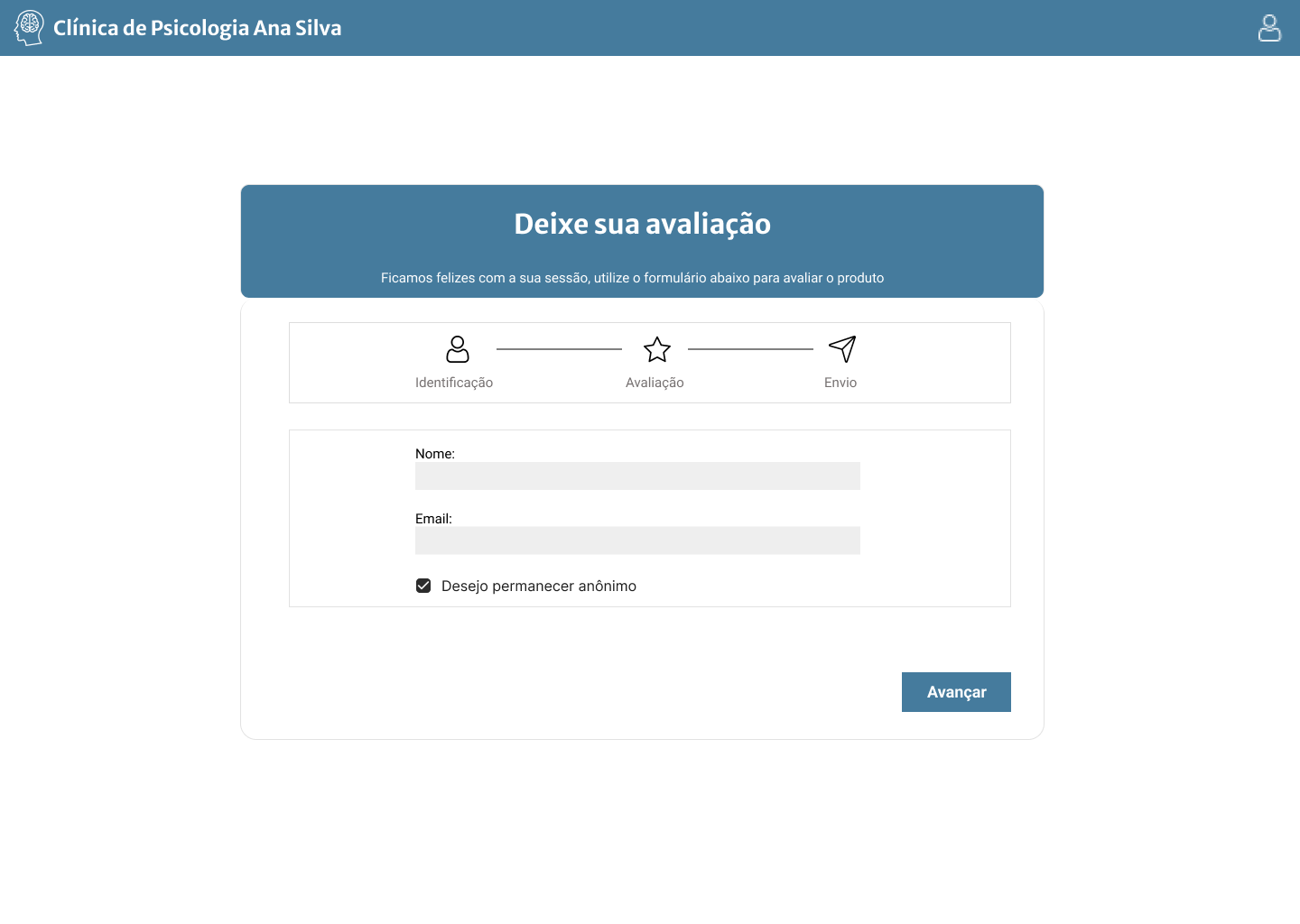
A interface do sistema deve ser intuitiva e fácil de usar, especialmente para usuários não técnicos, como a psicóloga e os pacientes.

1. Compatibilidade

O sistema deve ser compatível com os principais navegadores e dispositivos móveis, garantindo que tanto a psicóloga quanto os pacientes possam acessá-lo de qualquer dispositivo.

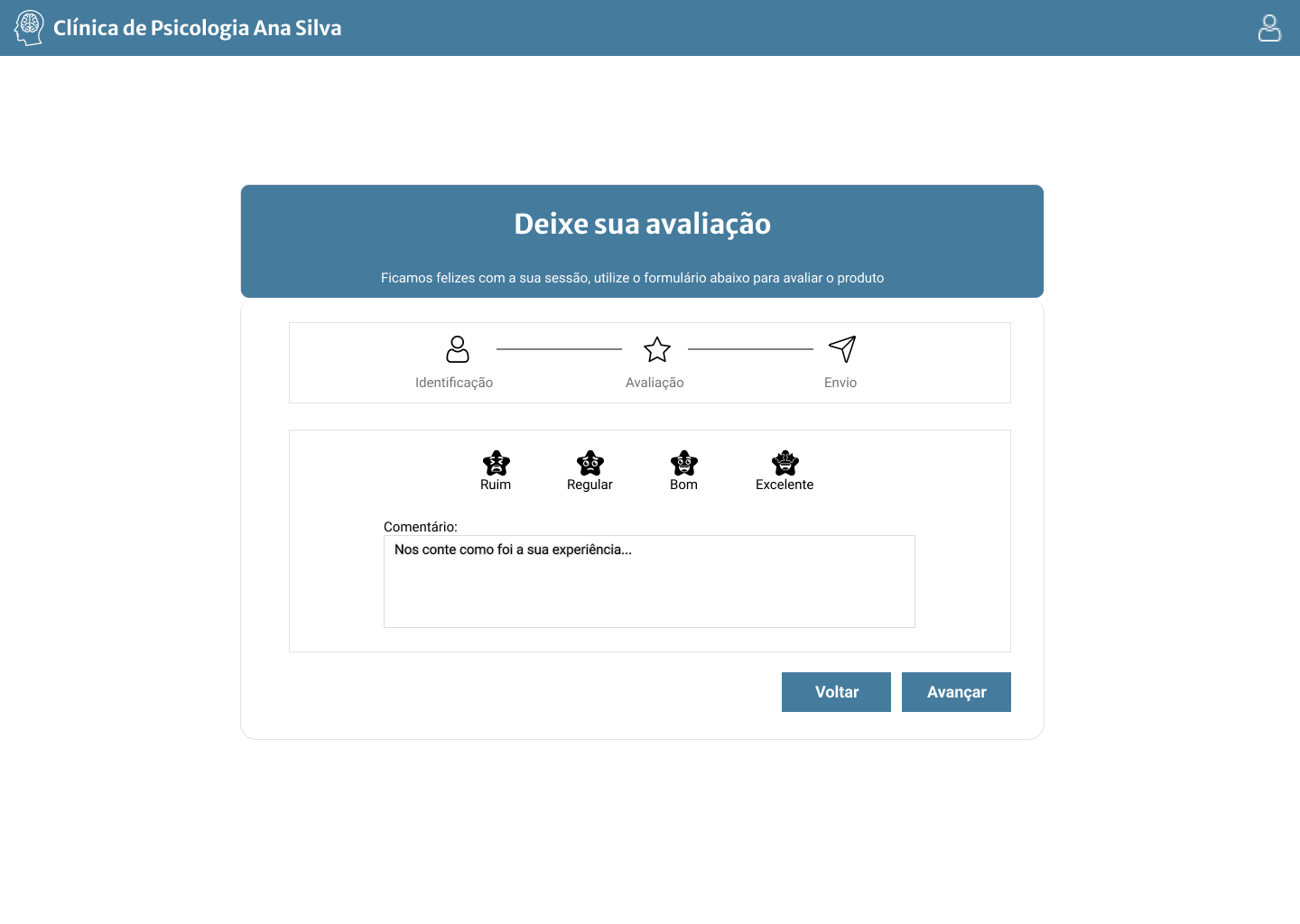
* 1. Protótipo

Tela 1: /



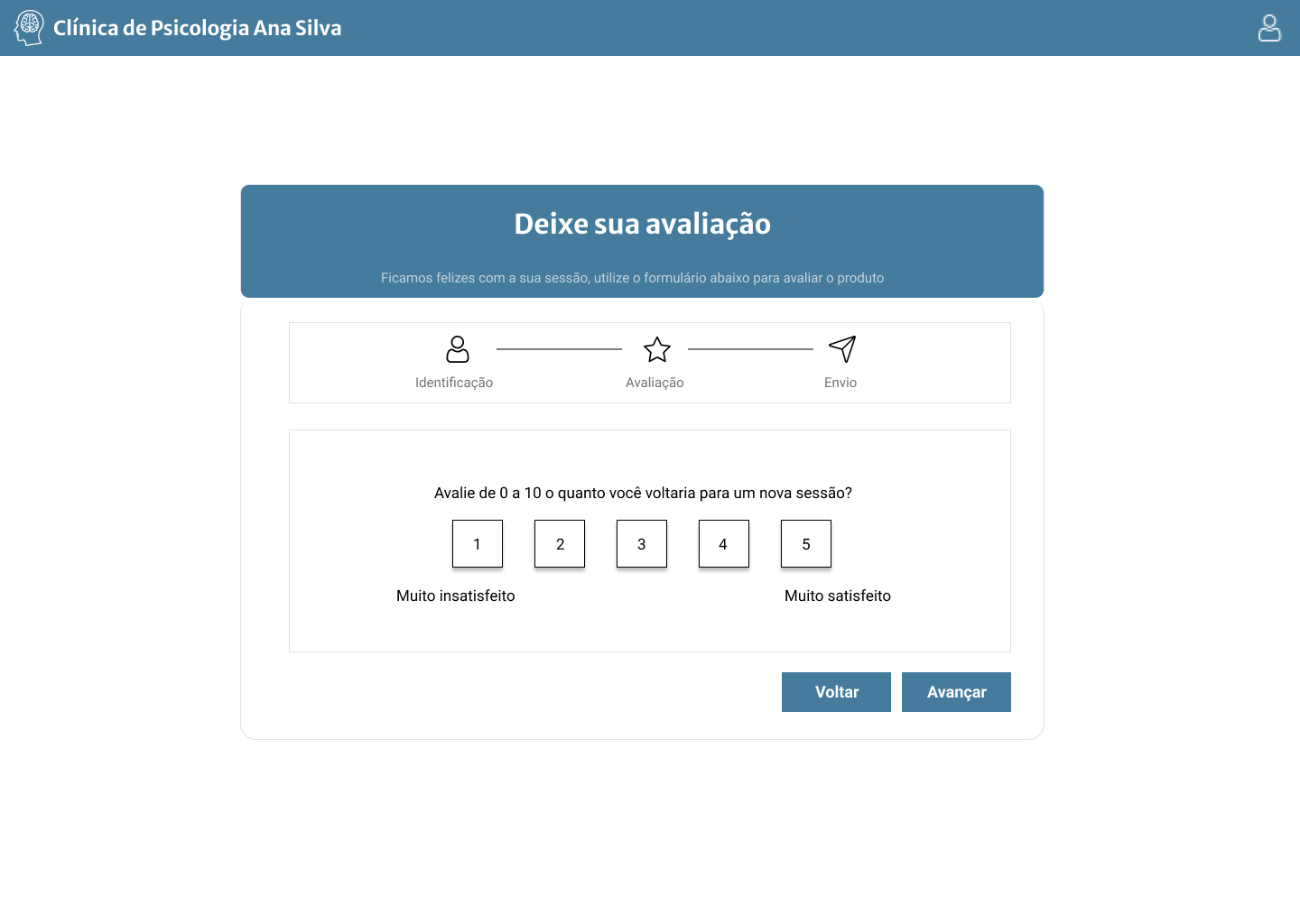
* Objetivo da tela: Permitir que o paciente escolha se identificar ou nâo
* De onde é chamada: Está e a tela incial ao acessar o sistema.
* Regras de Domínio:
  + Campo de e-mail: aceita apenas endereços de email válidos.
* Tipos de usuários: Psicóloga e pacientes podem acessar esta tela.

Tela 2: /avaliacao/avalie



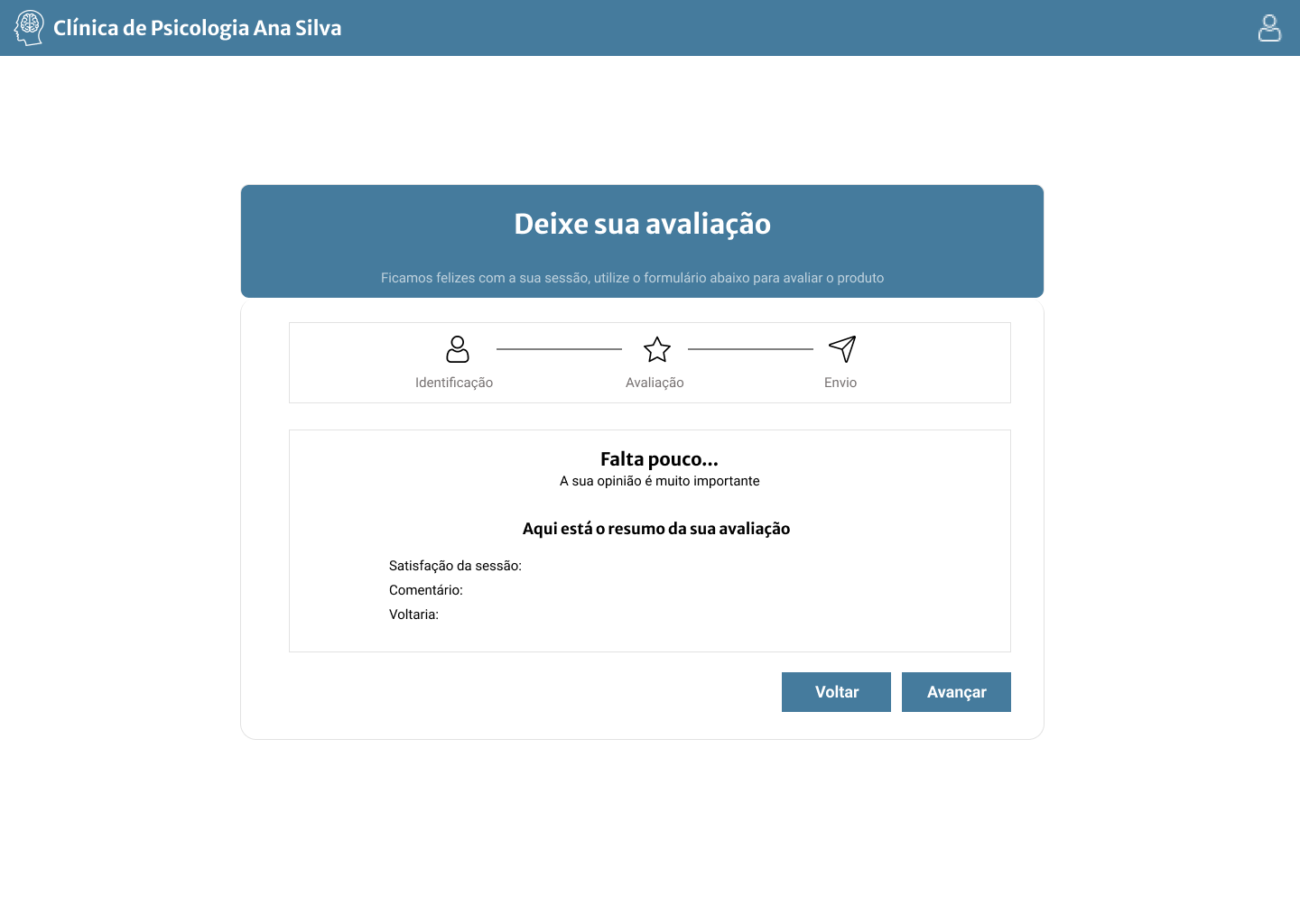
* Objetivo da tela: Permitir que o paciente avalie a sessão entre: ruim, regular, bom e excelente. Permitindo também que o mesmo possa fazer um comentário sobre a sessão.
* De onde é chamada: A partir que o usuário passa da página de identificação/home.
* Tipos de usuários: Psicóloga e pacientes podem acessar esta tela.

Tela 3: avaliacao/avalie/voltaria



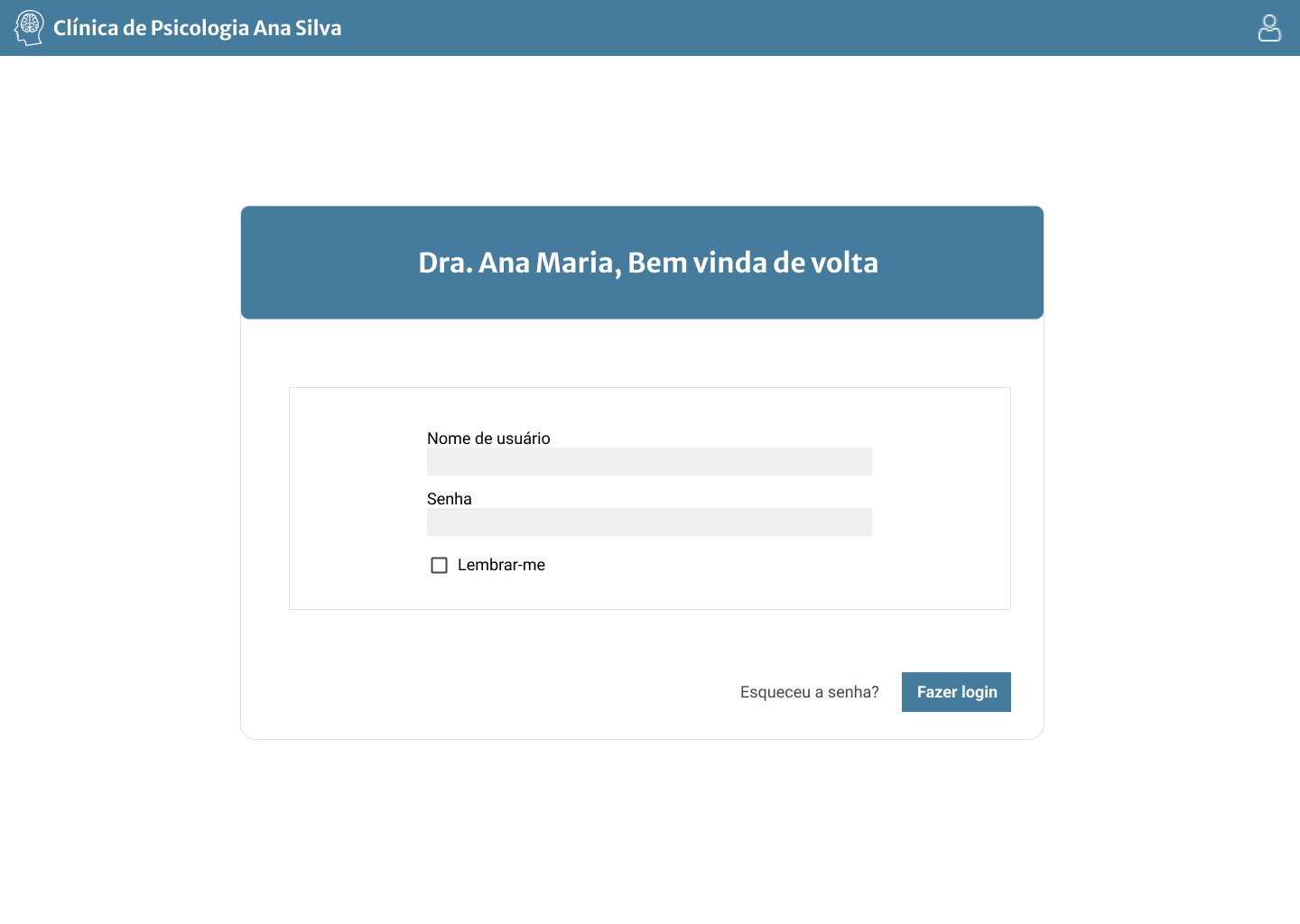
* Objetivo da Tela: Permtir que o paciente avalie de 1 a 5 o quanto ele voltaria para uma nova sessão.
* De onde é chamada: A partir que o paciente tenha avaliado a sessão.
* Tipos de usuários: Psicóloga e pacientes podem acessar esta tela.

Tela 4: /avaliacao/envio



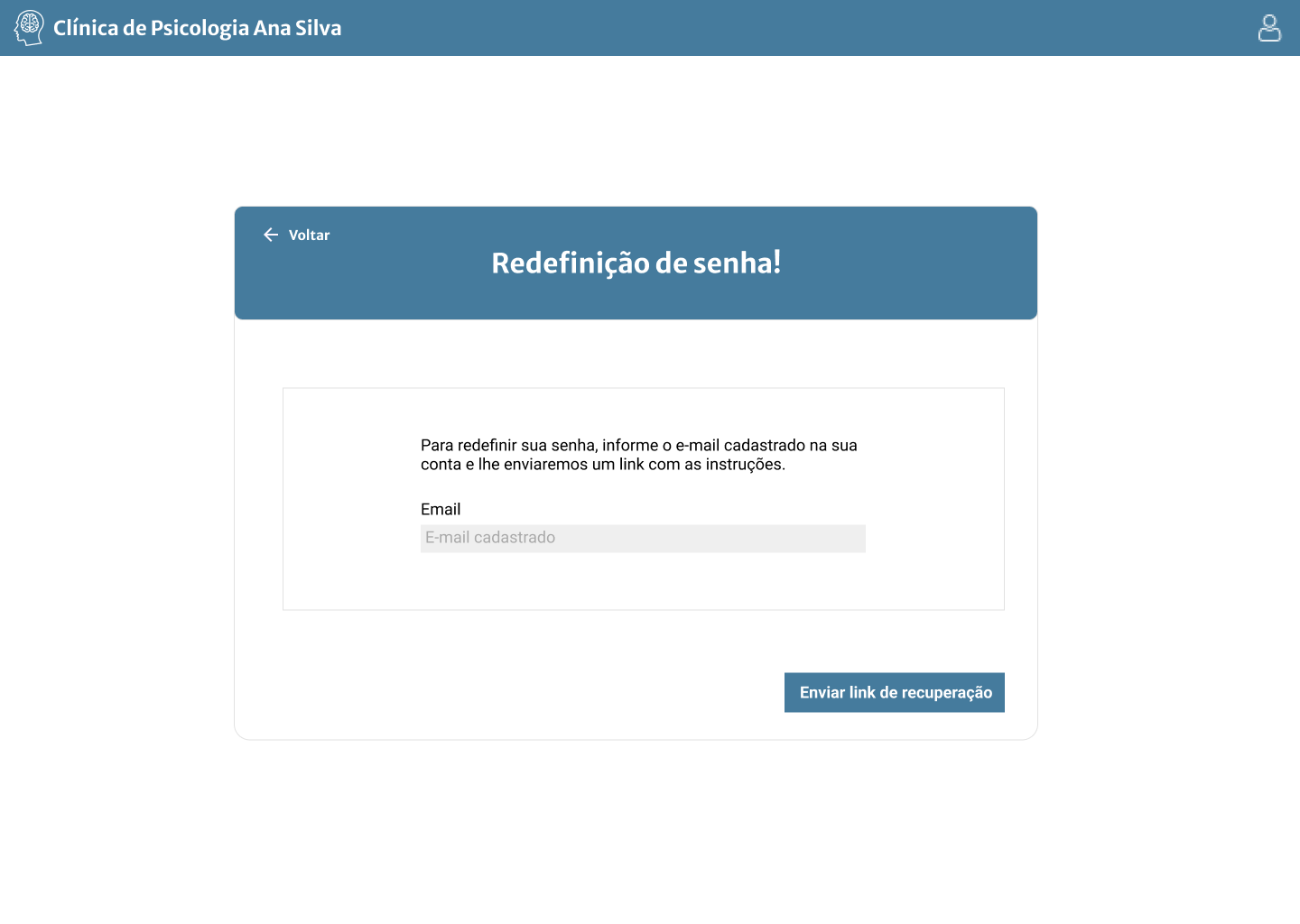
* Objetivo da Tela: Permitir que o paciente veja o resumo da avaliação dele como: satisfação da sessão, o comentário e o quanto ele voltaria para uma nova sessão.
* De onde é chamada: Assim que o usuário finaliza a avaliação da sessão.
* Tipos de usuários: Psicóloga e pacientes podem acessar esta tela.

Tela 5: /login



* Objetivo da Tela: Permitir que a psicóloga faça a autenticação e acesse sua conta no sistema
* De onde é chamada: A partir que a psicóloga clique no icone de usuário na header (cabecalho) na parte superior.
* Regras de Domínio:
  + Campo de nome de usuário: aceita apenas nomes de usuário válidos cadastrados no banco de dados.
  + Campo de senha: aceita apenas a senha vinculada ao nome de usuário no banco de dados.
* Tipos de Usuários: Psicóloga e pacientes podem acessar esta tela. No entanto, apenas a psicóloga pode se autenticar e acessar o sistema.

Tela 6: /login/forgotpassword

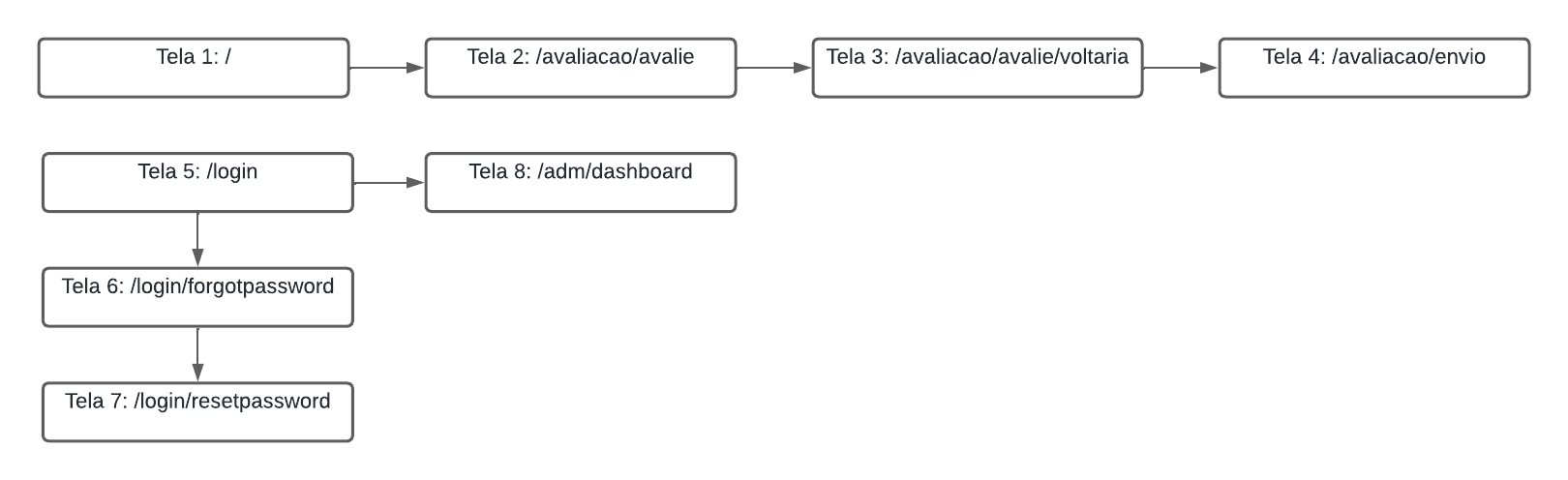


* Objetivo da Tela: Permitir que a psicóloga caso tenha esquecido sua senha ou por qualquer outro motivo, consiga enviar um e-mail com o link de recuperação de senha.
* De onde é chamada: A partir do momento em que a psicóloga clicar em esqueceu a senha na página de login.
* Regras de Domínio:
  + Campo de e-mail: Apenas e-mails cadastrados no banco de dados serão aceitos.
* Tipos de Usuários: Psicóloga e pacientes podem acessar esta tela. No entanto, apenas a psicóloga pode se autenticar e acessar o sistema.

Tela 7: /login/resetpassword

Tela 8: /adm

* Objetivo da Tela: Permitir que a psicóloga visualize os feedbacks fornecidos pelos pacientes em um dashboard.
* De onde é chamada: A partir do momento em que a psicóloga acesse sua conta no sistema.
* Tipos de Usuários: Somente a psicológa tem acesso a esta tela.
  + 1. **Diagrama de Navegação**



* 1. **Métricas e Cronogramas**

**Pontos de Casos de Uso**

* Login e autenticação: Complexidade baixa.
* Feedback de paciente: Complexidade média.
* Visualização de Feedback pela psicóloga: Complexidade alta.

Estimativa:

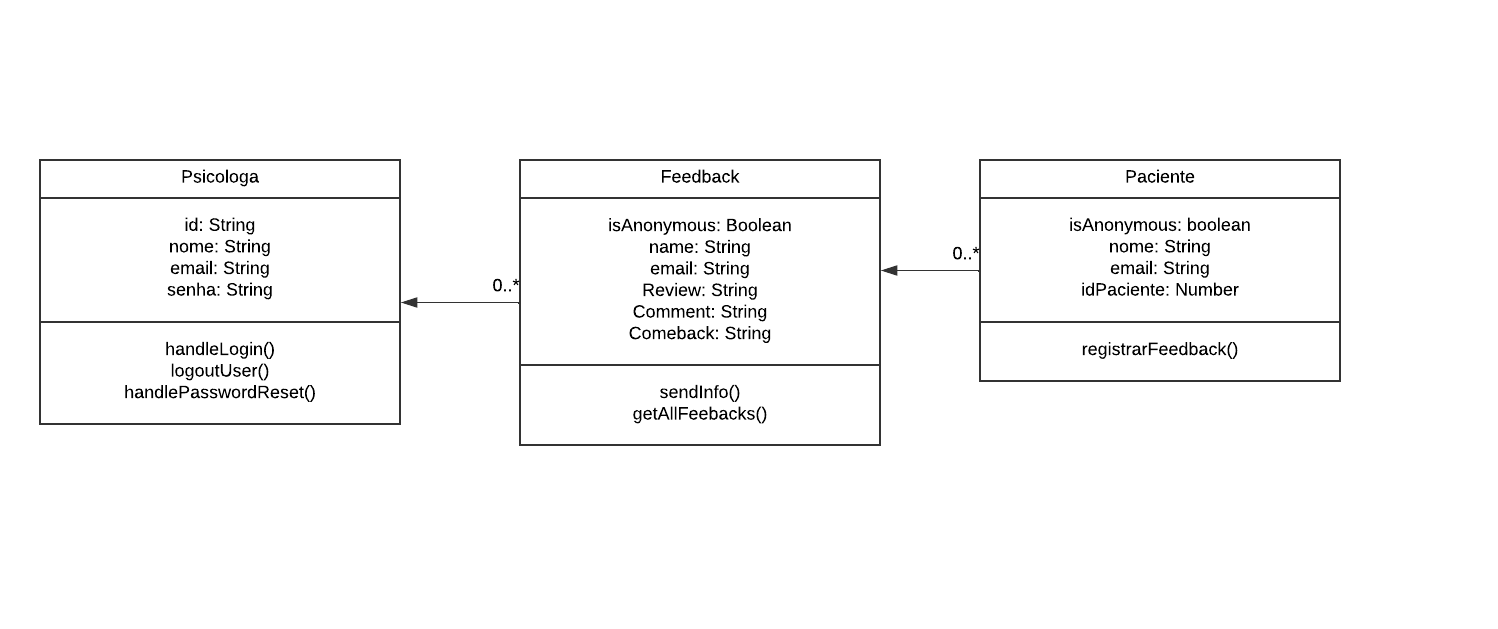
|  |  |  |
| --- | --- | --- |
| Funcionalidade | Pontos de Função | Estimativa de Esforço (em horas) |
| Autenticação (login/logout) | 10 PF | 8 Horas |
| Tela de Recuperação de Senha (esqueci senha) | 8 PF | 6 Horas |
| Tela de Redefinição de Senha | 8 PF | 6 Horas |
| Envio de feedback pelos pacientes | 15 PF | 12 Horas |
| Visualização de feedbacks | 20 PF | 16 Horas |
| Dashboard de dados | 25 PF | 20 Horas |

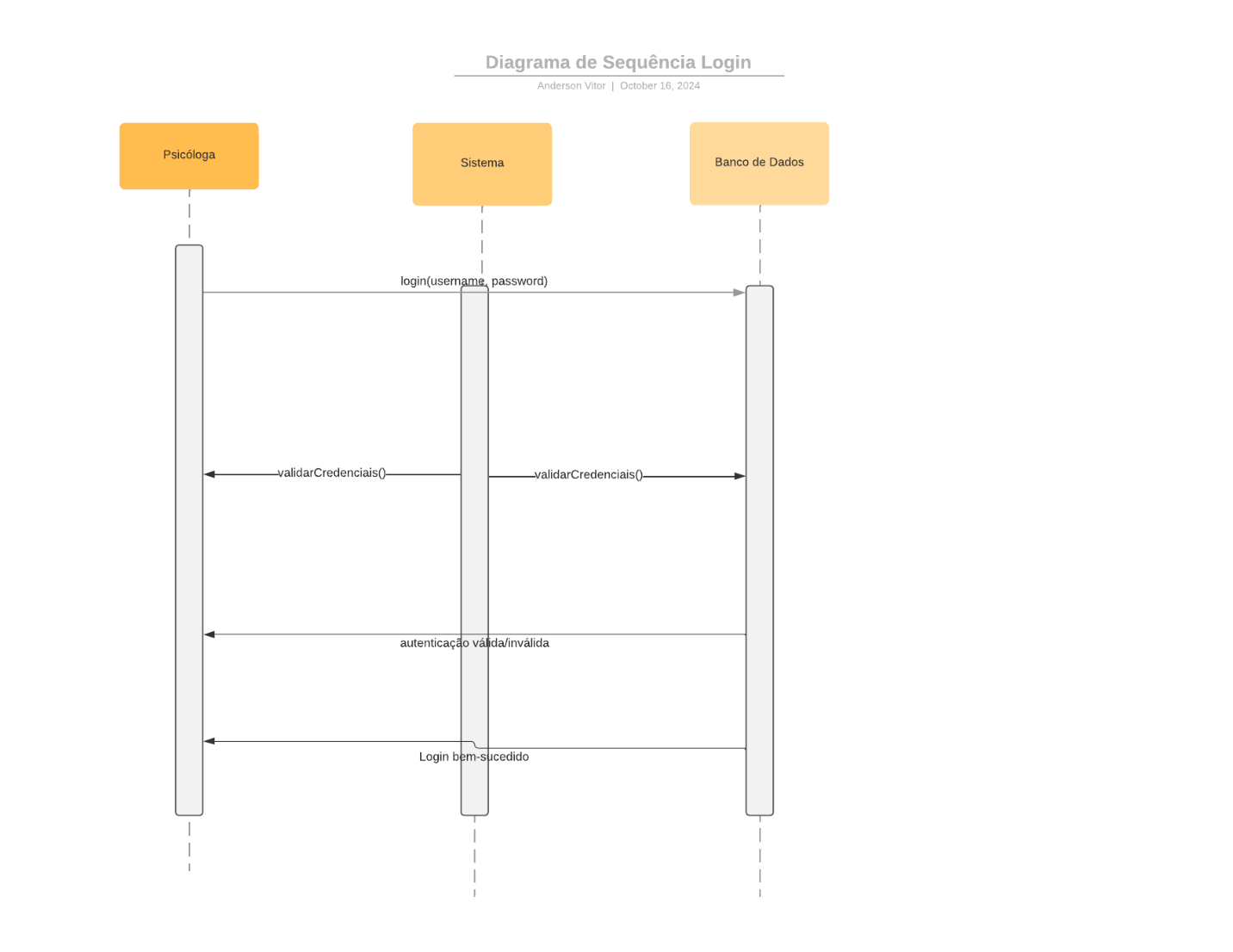
Métrica Completa

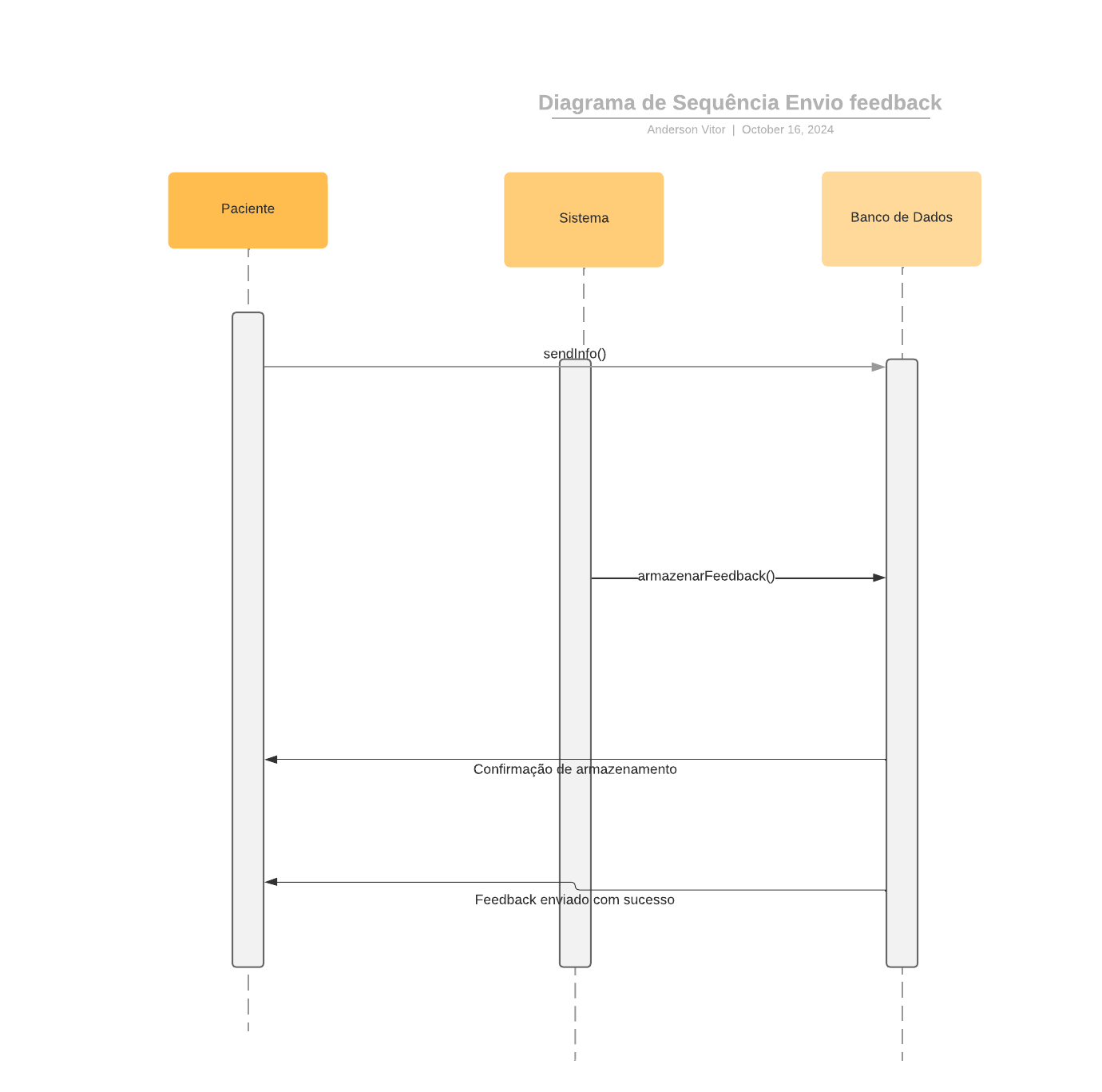
|  |  |
| --- | --- |
| Atividade | Estima de esforço (em horas) |
| Prototipagem | 10 Horas |
| Desenvolvimento | 54 Horas |
| Documentação | 10 horas |

1. **Análise e Design**
   1. **Arquitetura do Sistema**

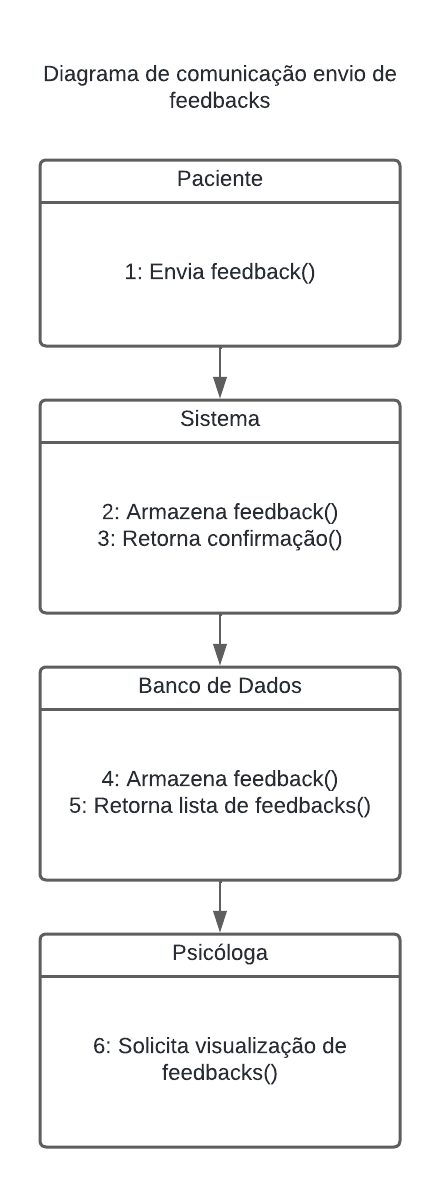
* Tipo de Arquitetura: Cliente/Servidor de 3 camadas (Frontend, Backend, Banco de Dados)
* Frontend: React.js para a interface com os pacientes e psicóloga.
* Backend: Node.js com Express para lidar com a autenticação e comunicação com o banco de dados.
* Banco de Dados: MongoDB para armazenar feedbacks, informações dos pacientes e dados da psicóloga.
* Configuração de Hardware: Requisitos mínimos para servidores de hospedagem, como capacidade de processamento e memória.
* Rede: Utilização de HTTPS para garantir a segurança na comunicação.
* Dimensionamento de Conexões: Conexões simultâneas esperadas (12 acessos diários)
  1. **Modelo do Domínio**

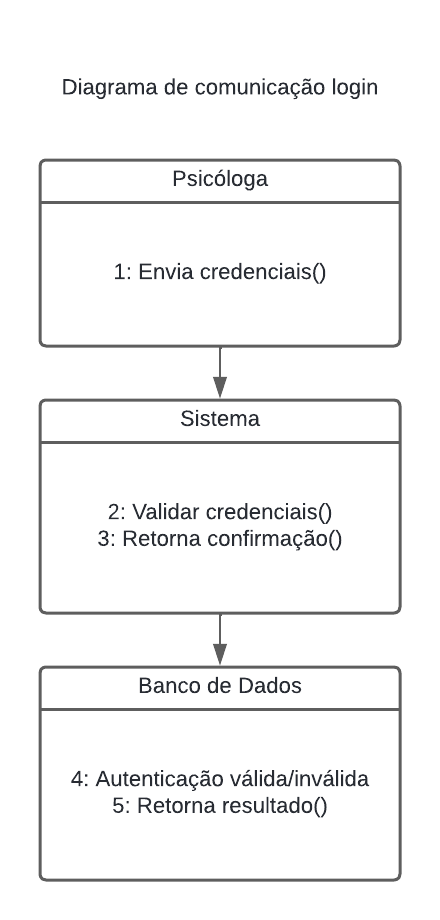


* 1. **Diagramas de Interação**
     1. **Diagrama de Sequência**

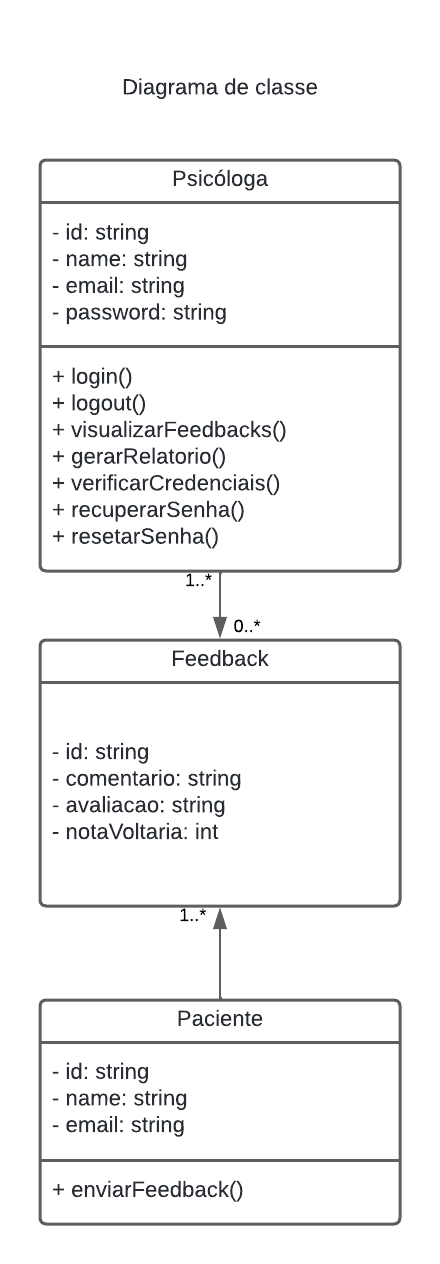


* + 1. **Diagrama de Colaboração/Comunicação**

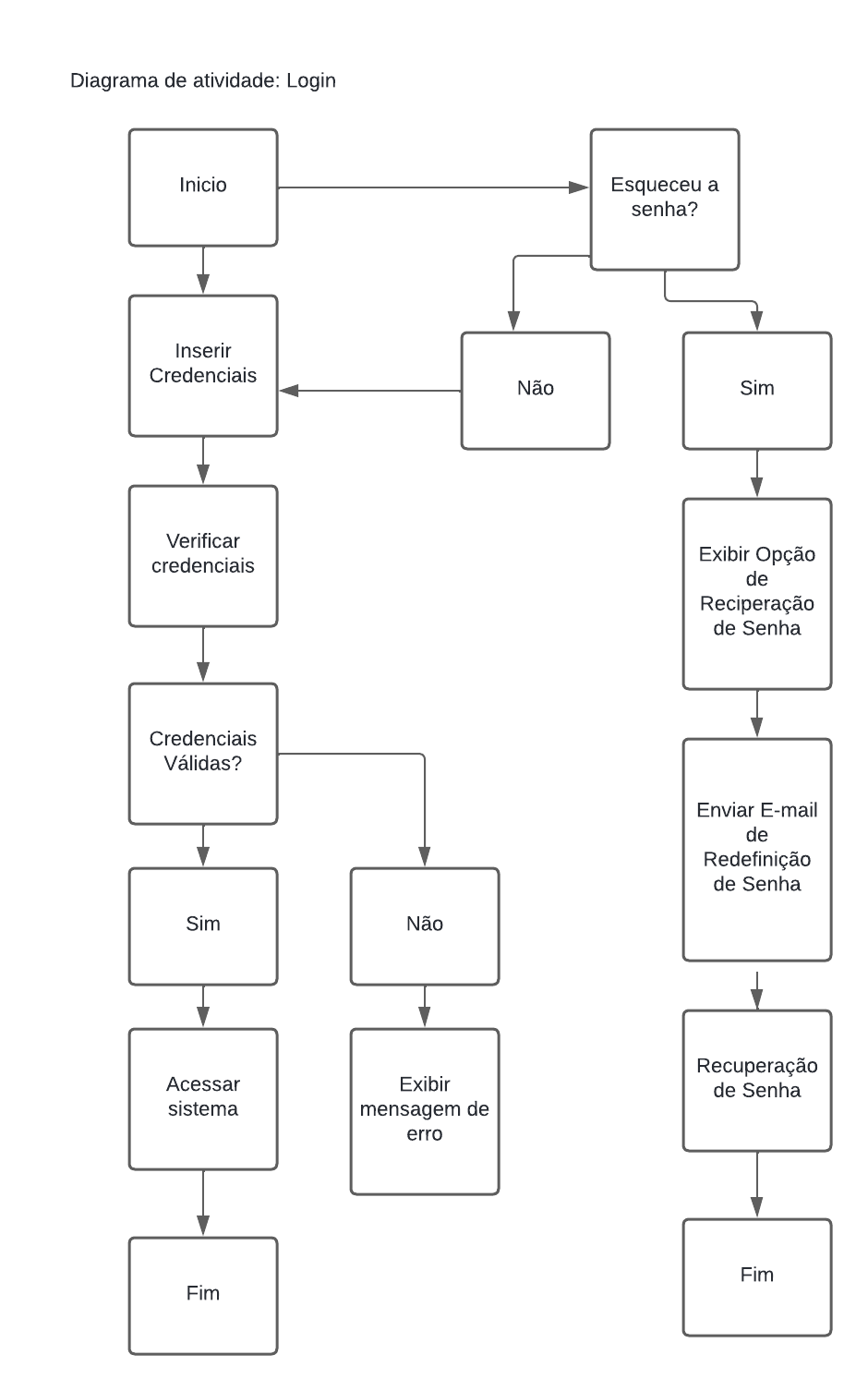




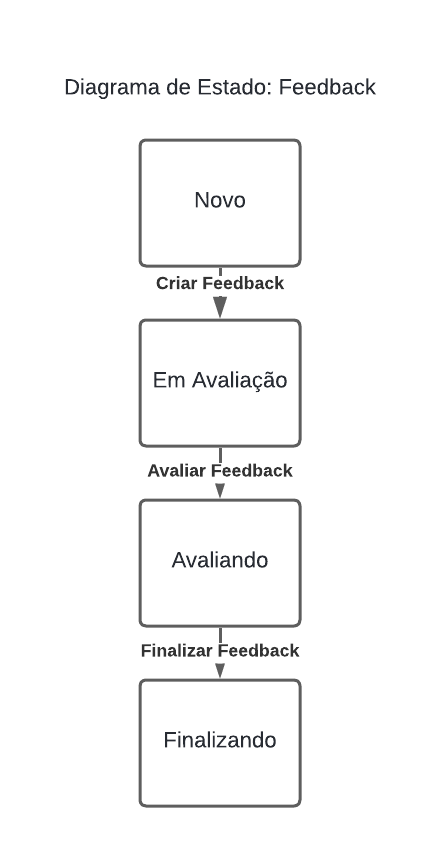
* 1. **Diagrama de Classe**

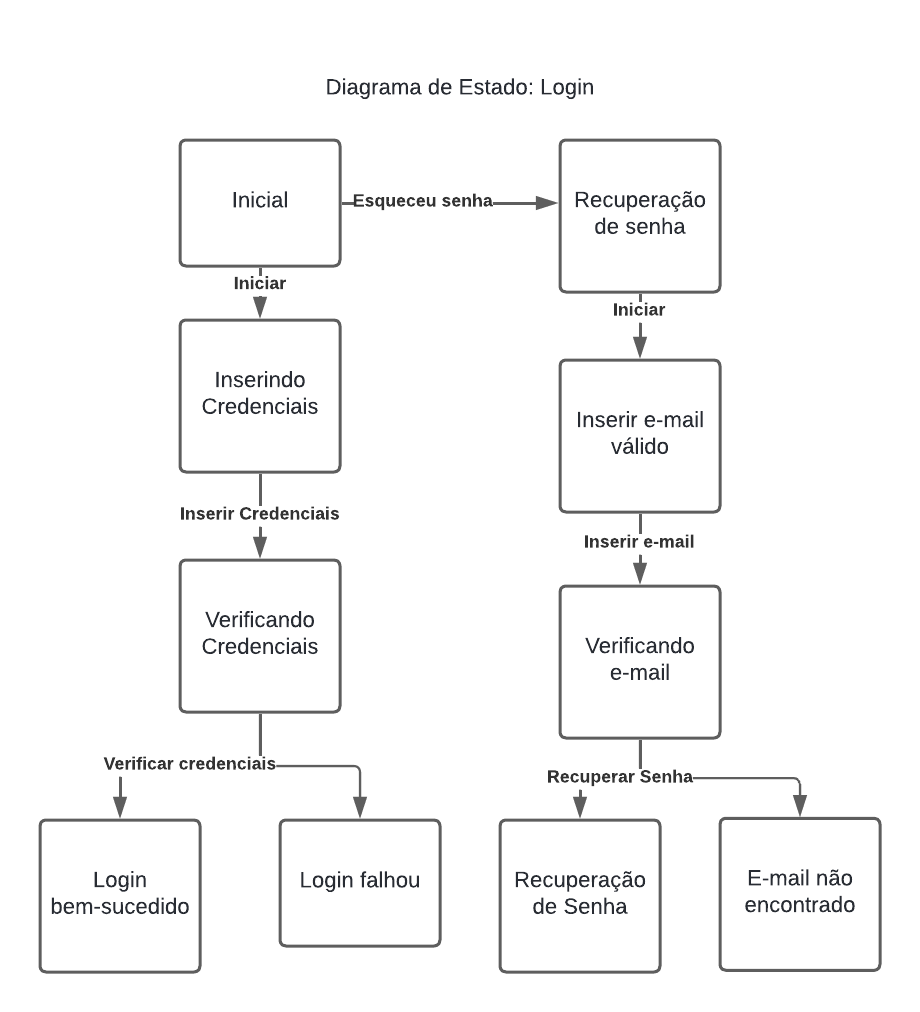


* 1. **Diagrama de Atividades**

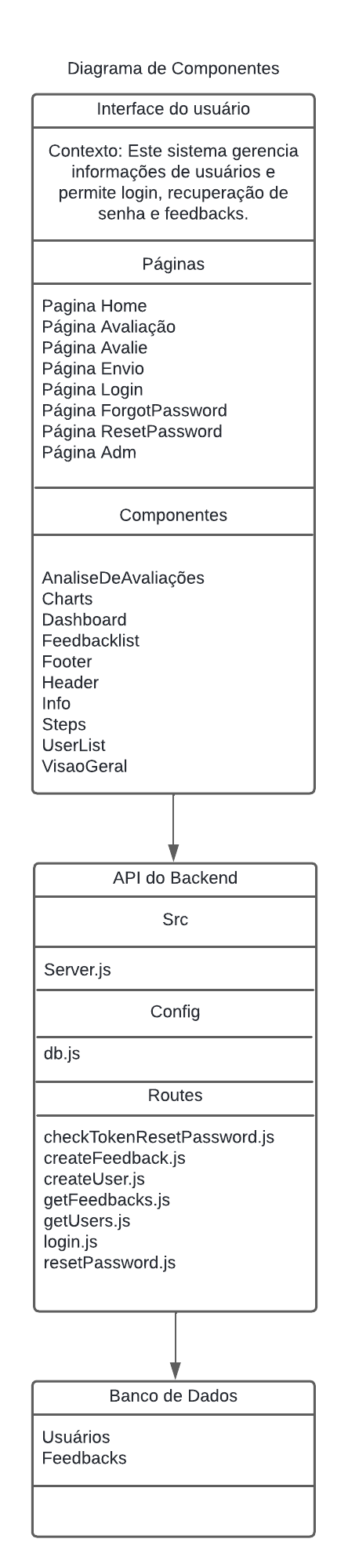


* 1. **Diagrama de Estados**





* 1. **Diagrama de Componentes**



* 1. **Modelo de Dados**
     1. **Modelo Lógico da Base de Dados**

1. Modelo Entidade-Relacionamento (ER)
   * Entidades: Psicóloga, Feedback e Paciente
   * Atributos:
     + Psicóloga: ID, username, e-mail, passsword.
     + Feedback: isAnonymous, id, name, email, review, comment, comeback
     + Paciente: id, name, email
   * Relacionamentos:
     + Um paciente pode ter vários feedbacks
     + Uma psicóloga pode gerenciar multiplos feedbacks
     1. **Criação Física do Modelo de Dados**

// CreateFeedback.js

const express = require('express')

const router = express.Router()

const Mongo = require('../config/db')

router.put('/', async (*req*, *res*) => {

    try {

        const userData = *req*.body

        console.log(userData)

        const db = await Mongo()

        await db.collection('feedbacks').insertOne(userData) // Adicionando um novo feedback

        console.log('Feedback Adicionado com sucesso')

*res*.status(201)

*res*.send(userData)

    } catch(error) {

        console.error(error)

    }

})

*module*.*exports* = router

* + 1. **Dicionário de Dados**

|  |  |  |  |
| --- | --- | --- | --- |
| Coleção | Atributos | Tipo | Descrição |
| Feedbacks | \_id | ObjectId | Identificador único do feedback |
|  | isAnonymous | Boolean | Anônimo ou Não |
|  | Name | String | Nome do paciente |
|  | Email | String | E-mail do paciente |
|  | Review | String | Avaliação do paciente |
|  | Comment | String | Comentário do paciente |
|  | Comeback | Number | Quanto o paciente voltária em uma próxima sessão |
| Usuários | \_id | ObjectId | Identificador único do usuário (psicóloga) |
|  | username | String | Nome do usuário (psicóloga) |
|  | Password | String | Senha do usuário (psicóloga) |
|  | Email | String | E-mail do usuário (psicóloga) |

* 1. **Ambiente de Desenvolvimento**

Softwares:

* Linguagem de Programação: JavaScript (Node.js e React)
* Backend: Node.js com Express.js
* Frontend: React.js
* Banco de Dados: MongoDB (banco de dados NoSQL)
* Gerenciamento de Pacotes: npm (Node Package Manager)
* Ferramentas de Desenvolvimento: Visual Studio Code (IDE), Git (controle de versão).
* Versionamento de Código: Git e GitHub para controle de versão e colaboração
* Bibliotecas e Frameworks:
  + Express.js: Framework web para Node.js
  + React.js: Biblioteca para contrução de interfaces de usuário (UI).
  + Styled-componentes: Estilização de componentes no React.
  + Mui/material: Estilização de componentes.
  + Recharts: Biblioteca para criação de gráficos em JavaScript.
  + React icons: Biblioteca de icones React.
  + React Admin: Biblioteca para criação de página administrativa.
  + Axios: Cliente HTTP para realizar requisições à API.
  + JWT (Json Web Token): Autenticação de usuários.
  + Bcrypt: hash de senhas.
  + React-toastify: Notificações no frontend
  + Cors: Transmissão de HTTP headers no backend.
  + dotEnv: Variáveis de ambiente.
* Equipamentos de Hardware:
  + Computador para Desenvolvimento:
    - Processador Ryzen 5 4600G
    - Memória RAM: 4GB ou superior
    - Armazenamento: 240GB SSD ou superior.
  + Rede:
    - Conexão à internet de alta velocidade para colaboração em Github, consultas à documentação online, downloads de dependências e comunicação com a equipe.
* Redes:
  + GitHub: Utilizado para hospedar o repositório de código-fonte e colaborar com outros desenvolvedores.
  + MongoDB Atlas: Solução de banco de dados em nuvem, permitindo acesso remoto ao banco de dados durante o desenvolvimento.
  + Vercel: Hospedagem de ambiente de desenvolvimento para APIs e frontend, facillitando a entega contínua e testes.
  1. **Sistemas e componentes externos utilizados**

* MongoDB Atlas: Banco de dados em nuvem para armazenar feedbacks, usuários e dados do sistema de forma segura e escalável.
* JWT (JSON Web Token): Componente utilizados para implementar a autenticação segura no sistema, permitindo que os usuários acessem suas contas e protejam suas sessões.
* Bcrypt: Utilizado para realizar o hash de senhas dos usuários antes de armazená-las no banco de dados, garantindo a segurança dos dados.
* Axios: Cliente HTTP que facilita a comunicação entre o frontend (React.js) e o backend (Node.js/Express).
* Styled-components: Componente externo para estilização de páginas no React, que facilita a criação de Uis dinâmicas e responsivas com CSS-in-JS.
* Recharts: Biblioteca de gráficos utilizada para exibir os dados de feedbacks de forma visual no sistema da psicóloga.
* **MUI (Material-UI)**: Biblioteca para componentes de interface.
* **Semantic Release**: Ferramenta para automação de versionamento semântico e publicação contínua, com base nas mensagens de commit.

1. **Implementação**

Server.js

// Server.js

\* @decription Arquivo principal do servidor

\* @author Anderson

const express = require('express')

const app = express()

const port = 2000

const cors = require('cors')

require('dotenv').config();

app.use(express.json())

// Precisamos expor o cabecalho para o react admin

app.use(cors({exposedHeaders: ['X-Total-Count']}))

// Importando as rotas

const createFeedBack = require('./Routes/createFeedBack')

const login = require('./Routes/login')

const getUsers = require('./Routes/getUsers')

const getFeedbacks = require('./Routes/getFeedbacks')

const resetPassword = require('./Routes/resetPassword')

const checkTokenResetPassword = require('./Routes/checkTokenResetPassword')

// Usando as rotas

app.use('/createFeedBack', createFeedBack)

app.use('/login', login)

app.use('/users', getUsers)

app.use('/feedbacks', getFeedbacks)

app.use('/request-password-reset', resetPassword)

app.use('/reset-password', checkTokenResetPassword)

// Iniciando o servidor

app.listen(port, () => {

    console.log('Servidor rodando na porta', port)

})

createFeedback.js

// Rota para adicionar feedbacks

\* @decription Rota para adicionar feedbacks no banco de dados

\* @author Anderson

const express = require('express')

const router = express.Router()

const Mongo = require('../config/db')

router.put('/', async (*req*, *res*) => {

    try {

        const userData = *req*.body

        console.log(userData)

// Chamando o arquivo do banco de dados

        const db = await Mongo()

        await db.collection('feedbacks').insertOne(userData) // Adicionando um novo feedback

        console.log('Feedback Adicionado com sucesso')

*res*.status(201)

*res*.send(userData)

    } catch(error) {

        console.error(error)

    }

})

// Exportando a rota para o servidor

*module*.*exports* = router

login.js

// Rota para login

\* @decription Rota para a autenticação da psicóloga

\* @author Anderson

const express = require('express')

const router = express.Router()

const jwt = require('jsonwebtoken')

const Mongo = require('../config/db')

const bcrypt = require('bcrypt')

const secretKey = 'minhasenha'

router.post('/', async (*req*, *res*) => {

    try {

        //Pegamos oque vem da requisição do frontend

        const {loginData} = *req*.body

        const db = await Mongo()

        const collection = await db.collection('usuarios')

        // Procurando usuário pelo nome fornecido

        const user = await collection.findOne({username: loginData.username})

        // Comparando senha fornecida com o hash armazenado no database

        const isPasswordCorrect = await bcrypt.compare(loginData.password, user.password)

        if (isPasswordCorrect) {

            const token = jwt.sign({name: user.username}, secretKey, {expiresIn: "1hr"})

            const userWithToken = {...user, token}

            return *res*.status(200).json(userWithToken)

        } else {

            return *res*.status(401).json({message: "Invalid credentials"})

        }

    } catch (error) {

        console.error(error)

    }

})

*module*.*exports* = router

resetPassword.js

// Rota para resetar a senha

\* @decription Rota para resetar a senha

\* @author Anderson

const express = require('express')

const router = express.Router()

const Mongo = require('../config/db')

const jwt = require('jsonwebtoken')

const nodemailer = require('nodemailer')

const senha = 'minhasenha'

router.post('/', async (*req*, *res*) => {

    const {email} = *req*.body

    console.log(email)

    try {

        const db = await Mongo()

        const collection = await db.collection('usuarios').find({}).toArray()

        const foundUser = collection.find(*user* => *user*.email === email)

        console.log(foundUser)

        if (!foundUser) {

            return *res*.status(400).send({ message: 'Usuário não encontrado' })

        }

        // gerando um token de recuperacao de senha

        const token = jwt.sign({id: foundUser.\_id}, senha, {expiresIn: '1hr'})

        // Configurando o transporte de email

        const transporter = nodemailer.createTransport({

            service: 'gmail',

            auth: {

                user: process.env.EMAIL\_USER,

                pass: process.env.EMAIL\_PASS

            },

        })

        // Enviando o email com o token

        const mailOptions = {

            from: process.env.EMAIL\_USER,

            to: foundUser.email,

            subject: 'Redefinição de senha',

            // Utilizando html aqui ao inves de text para deixar o email personalizavel

            html: `

            <div style="font-family: Arial, sans-serif; color: #333;">

                <h2 style="color: #2E86C1;">Olá, ${foundUser.username}!</h2>

                <p>Recebemos uma solicitação para redefinir sua senha.</p>

                <p>Para continuar com o processo de redefinição, clique no botão abaixo:</p>

                <a href="http://localhost:3000/#/login/resetpassword?token=${token}"

                   style="display: inline-block; background-color: #28a745; color: white; padding: 10px 20px; text-decoration: none; border-radius: 5px;">

                   Redefinir Senha

                </a>

                <p style="margin-top: 20px;">Se você não solicitou essa alteração, ignore este e-mail.</p>

                <p>Atenciosamente,<br/>Equipe de Suporte</p>

                <div style="margin-top: 30px; font-size: 12px; color: #999;">

                    <p>Este é um e-mail automático, por favor não responda.</p>

                </div>

            </div>

        `,

        }

        try {

            await transporter.sendMail(mailOptions)

*res*.status(200).json(token)

        } catch (error) {

            console.error('Erro ao enviar e-mail:', error) // Adicione esta linha

*res*.status(500).json({message: 'Erro ao enviar e-mail', error})

        }

    } catch (error) {

        console.error(error)

    }

})

*module*.*exports* = router

checkTokenResetPassword.js

// Rota para checar se o token para resetar a senha está correto

\* @decription Rota para verificar o token

\* @author Anderson

const express = require('express')

const router = express.Router()

const Mongo = require('../config/db')

const jwt = require('jsonwebtoken')

const { ObjectId } = require('mongodb')

const bcrypt = require('bcrypt')

const saltRounds = 12

const senha = 'minhasenha'

router.post('/:token', async (*req*, *res*) => {

    const { token } = *req*.params

    const { newPassword } = *req*.body

    console.log(token, newPassword)

    // Criptografando a nova senha

    bcrypt.hash(newPassword, saltRounds, async (*err*, *hash*) => {

        if (*err*) throw *err*

        console.log('Senha criptografada', *hash*)

        try {

            const db = await Mongo()

            const decoded = jwt.verify(token, senha)

            const foundUser = await db.collection('usuarios').findOne({ \_id: **new** ObjectId(decoded.id) })

            console.log(foundUser)

            if (!foundUser) {

                return *res*.status(400).send({ message: 'Token inválido' })

            }

            await db.collection('usuarios').updateOne(

                { \_id: foundUser.\_id },

                { $set: { password: *hash* } }

            )

            console.log('Senha atualizada com sucesso')

*res*.status(200).send({ message: 'Senha atualizada com sucesso' })

        } catch (error) {

            console.error(error)

        }

    })

})

*module*.*exports* = router

Frontend

Context.jsx

// Contexto

\* @decription Arquivo de contexto para salvar info

\* @author Anderson

import React, { createContext, useEffect, useState } from "react";

import axios from 'axios'

import { toast } from "react-toastify";

import { useNavigate } from "react-router-dom";

import {jwtDecode} from "jwt-decode"

export const Context = createContext(null)

export const ContextProvider = ({ *children*, *formComponents* }) => {

    const [isChecked, setIsChecked] = useState(false)

    const [isRequired, setIsRequired] = useState(true)

    const [isReadOnly, setIsReadOnly] = useState(false)

    const [data, setData] = useState({ isAnonymous: isChecked, name: "", email: "", review: "", comment: "", comeback: "" })

    const [loginData, setLoginData] = useState(

        JSON.parse(localStorage.getItem("@:user")) || JSON.parse(sessionStorage.getItem("@:user")) || { username: "", password: "" }

    );

    const [isLembrarMe, setIsLembrarMe] = useState(false)

    const [currentStep, setCurrentStep] = useState(0)

    const [allFeedbacks, setAllFeedbacks] = useState({})

    const navigate = useNavigate()

    // Lógica no botao para avançar e mudar o currentStep

    const changeStep = (*i*, *e*) => {

        // Se i for menor que 0 ou maior ou igual ao tamanho do array

        if (*i* < 0 || *i* >= *formComponents*.length) return // se for fora dos limites nao faz nada

        // se I é um indice válido, atualiza o estado 'currentStep' com o novo indice

        setCurrentStep(*i*)

    }

    // Logica para enviar as informações para o backend

    const sendInfo = async (*e*) => {

*e*.preventDefault()

        try {

            const response = await axios.put("http://localhost:2000/createFeedBack", data, {

                headers: { 'Content-Type': 'application/json' }

            })

            if (response.status === 201) {

                toast.success('Feedback enviado com sucesso')

                navigate('/')

                setCurrentStep(0)

                setData({ name: "", email: "", review: "", comment: "", comeback: "" })

            } else {

                toast.error('Ocorreu algum erro, tente novamente em alguns instantes')

            }

        } catch (error) {

            console.error(error)

        }

    }

    // Pegar todos os feedbacks para criar um dashBoard

    const getAllFeedBacks = async () => {

        try {

            const response = await axios.get("http://localhost:2000/feedbacks")

            setAllFeedbacks(response.data)

            return response.data

        } catch (error) {

            console.log(error)

        }

    }

    // Funcao para login na pagina de adm da psicologa

    const handleLogin = async (*e*) => {

*e*.preventDefault()

        if (loginData.username === "" && loginData.password === "") {

            toast.warning('Preencha as informações')

        }

        try {

            const response = await axios.post('http://localhost:2000/login', { loginData },

                { headers: { 'Content-Type': 'application/json' } })

            if (response.status === 200) {

                const token = response.data.token

                axios.defaults.headers.common["Authorization"] = `Bearer ${token}`

                localStorage.setItem("@:token", token)

                if (isLembrarMe) {

                    localStorage.setItem("@:user", JSON.stringify(loginData));

                    sessionStorage.setItem("@:user", JSON.stringify(loginData));

                    localStorage.setItem("@:token", token)

                    sessionStorage.setItem("@:token", token)

                }

                navigate("/adm")

                getAllFeedBacks()

            }

        } catch (error) {

            if (error.response?.status === 401) {

                toast.error('Invalid credentials')

            } else {

                toast.error('An error has occurred')

            }

        }

    }

    // Function to logout user from admin page

    const logOutUser = () => {

        // limpando das memorias

        sessionStorage.removeItem("@:user")

        sessionStorage.removeItem("@:token")

        localStorage.removeItem("@:user")

        localStorage.removeItem("@:token")

        // tirando da header

        axios.defaults.headers.common["Authorization"] = '';

        // limpando dos estados

        setLoginData({username: '', password: ""})

        //Redirecionando

        navigate('/')

        toast.info("Você foi desconectado por motivos de segurança")

    }

    // Function to check if the token is still available

    const checkTokenExpiracy = () => {

        const token = localStorage.getItem("@:token")

        if (token) {

          const decoded = jwtDecode(token)

          const currentTime = *Date*.now() / 1000

          if (decoded.exp < currentTime) {

            console.log("Token expirado, deslogando...")

            logOutUser()

          }

        }

    }

    // Now using useEffect to check if tokenexpiracy

    useEffect(() => {

        checkTokenExpiracy()

        // Verificando a expiração

        const intervalCheck = setInterval(checkTokenExpiracy, 60000)

        return () => clearInterval(intervalCheck)

    })

    const contextValue = {

        data,

        setData,

        isChecked,

        setIsChecked,

        isRequired,

        setIsRequired,

        currentStep,

        setCurrentStep,

        changeStep,

        sendInfo,

        isReadOnly,

        setIsReadOnly,

        handleLogin,

        loginData,

        setLoginData,

        isLembrarMe,

        setIsLembrarMe,

        allFeedbacks,

        setAllFeedbacks,

        getAllFeedBacks,

        navigate

    }

    return (

        <*Context.Provider* *value*={contextValue}>

            {children}

        </*Context.Provider*>

    )

}

Login.jsx

// Página para autenticação da psicologa

\* @decription Página de autenticação frontend

\* @author Anderson

import React, { useContext } from 'react'

import \* as Home from '../Home/Styles'

import \* as L from './Styles'

import { Context } from '../../Context/Context'

export const Login = () => {

    const {handleLogin, loginData, setLoginData, isLembrarMe, setIsLembrarMe, navigate} = useContext(Context)

    return (

        <*Home.Content*>

            <*L.Info*>

                <h1>Dra. Ana Maria, Bem vinda de volta</h1>

            </*L.Info*>

            <*Home.FormContainer* *className*='login'>

                <*Home.inputContainer*>

                    <*Home.labelContent*>

                        <label *htmlFor*="user">Nome de usuário:</label>

                        <input *type*="text" *name*="user" *id*="user" *value*={loginData.username || ''} *onChange*={(*e*) => setLoginData((*prev*) => ({...*prev*, username: *e*.target.value}))} />

                    </*Home.labelContent*>

                    <*Home.labelContent*>

                        <label *htmlFor*="password">Senha:</label>

                        <input *type*="password" *name*="password" *id*="password" *value*={loginData.password || ''} *onChange*={(*e*) => setLoginData((*prev*) => ({...*prev*, password: *e*.target.value}))} />

                    </*Home.labelContent*>

                    <*Home.checkContent*>

                        <input *type*="checkbox" *name*="lembrarme" *id*="lembrarme" *value*={isLembrarMe} *onChange*={(*e*) => setIsLembrarMe(*prevState* => !*prevState*)} />

                        <label *htmlFor*="check">Lembrar-me</label>

                    </*Home.checkContent*>

                </*Home.inputContainer*>

                <*Home.buttonContent*>

                    <p *onClick*={() => navigate('/login/forgotpassword')}>Esqueceu a senha?</p>

                    <*Home.buttonAvancar* *type*='submit' *onClick*={(*e*) => handleLogin(*e*)}>Fazer login</*Home.buttonAvancar*>

                </*Home.buttonContent*>

            </*Home.FormContainer*>

        </*Home.Content*>

    )

}

ForgotPassword.jsx

// Página de esqueceu a senha

\* @decription Página de esqueceu a senha

\* @author Anderson

import React, { useContext, useState } from 'react'

import \* as Home from '../Home/Styles'

import \* as F from './Styles'

import { FaArrowLeft } from "react-icons/fa";

import { Context } from '../../Context/Context';

import { useParams } from 'react-router-dom';

import axios from 'axios';

import { toast } from 'react-toastify';

export const ForgotPassword = () => {

    const {navigate} = useContext(Context)

    const {token} = useParams()

    const [email, setEmail] = useState('')

    const handlePasswordReset = async (*e*) => {

*e*.preventDefault()

        try {

            const response = await axios.post('http://localhost:2000/request-password-reset', {email})

            toast.success('E-mail enviado com sucesso')

        } catch (error) {

            if (error.status === 400) {

                toast.error('Usuário não encontrado')

            }

        }

    }

  return (

    <*Home.Content*>

        <*F.Info*>

            <span *onClick*={() => navigate('/login')}>

                <*FaArrowLeft*/>

                <p>Voltar</p>

            </span>

            <h1>Redefinicao de senha!</h1>

        </*F.Info*>

        <*Home.FormContainer*>

            <*Home.inputContainer*>

                <*Home.labelContent*>

                    <p>Para redefinir sua senha, informe o e-mail cadastrado na sua conta e lhe enviaremos um link com as instruções.</p>

                </*Home.labelContent*>

                <*Home.labelContent*>

                    <label *htmlFor*="email">Email:</label>

                    <input *type*="email" *name*="email" *id*="email" *placeholder*='E-mail cadastrado' *value*={email} *onChange*={(*e*) => setEmail(*e*.target.value)}/>

                </*Home.labelContent*>

            </*Home.inputContainer*>

            <*Home.buttonContent*>

                <*Home.buttonAvancar* *style*={{width: '244px'}} *onClick*={(*e*) => handlePasswordReset(*e*)}>Enviar link de recuperação</*Home.buttonAvancar*>

            </*Home.buttonContent*>

        </*Home.FormContainer*>

    </*Home.Content*>

  )

}

ResetPassword.jsx

// Página para resetar a senha

\* @decription Página para resetar a senha

\* @author Anderson

import React, { useContext, useState } from 'react'

import \* as Home from '../Home/Styles'

import \* as R from '../ResetPassword/Styles'

import { FaArrowLeft } from "react-icons/fa";

import { Context } from '../../Context/Context';

import { useLocation, useParams } from 'react-router-dom';

import axios from 'axios';

import { toast } from 'react-toastify';

export const ResetPassword = () => {

    const {navigate} = useContext(Context)

    // Pegando o token dos parametros

    const location = useLocation()

    const queryParams = **new** *URLSearchParams*(location.search)

    const token = queryParams.get('token')

    const [newPassword, setNewPassword] = useState('')

    const handleChangePaswword = async (*e*) => {

*e*.preventDefault()

        try {

            const response = await axios.post(`http://localhost:2000/reset-password/${token}`, {newPassword})

            toast.success('Senha atualizada com sucesso')

            navigate('/')

        } catch (error) {

            console.error(error)

        }

    }

  return (

    <*Home.Content*>

        <*R.Info*>

            <span *onClick*={() => navigate('/login')}>

                <*FaArrowLeft*/>

                <p>Voltar</p>

            </span>

            <h1>Redefinicao de senha!</h1>

        </*R.Info*>

        <*Home.FormContainer*>

            <*Home.inputContainer*>

                <*Home.labelContent*>

                    <p>Por favor, digite sua nova senha para completar o processo de redefinição.</p>

                </*Home.labelContent*>

                <*Home.labelContent*>

                    <label *htmlFor*="password">Nova senha:</label>

                    <input *type*="password" *name*="password" *id*="passwordreset" *placeholder*='Digite a nova senha' *value*={newPassword} *onChange*={(*e*)=> setNewPassword(*e*.target.value)} />

                </*Home.labelContent*>

            </*Home.inputContainer*>

            <*Home.buttonContent*>

                <*Home.buttonAvancar* *onClick*={(*e*) => handleChangePaswword(*e*)}>Redefinir</*Home.buttonAvancar*>

            </*Home.buttonContent*>

        </*Home.FormContainer*>

    </*Home.Content*>

  )

}

Envio.jsx

// Página de envio do feedback

\* @decription Página para enviar a avaliação

\* @author Anderson

import React, { useContext } from 'react'

import \* as Home from '../Home/Styles'

import { Info } from '../../Components/Info/Info'

import { Steps } from '../../Components/Steps/Steps'

import { Context } from '../../Context/Context'

import \* as E from './Styles'

import { BsFillEmojiFrownFill, BsFillEmojiHeartEyesFill, BsFillEmojiNeutralFill, BsFillEmojiSmileFill } from 'react-icons/bs'

export const Envio = () => {

    const { currentStep, changeStep, data, sendInfo, navigate } = useContext(Context)

    const previousPage = (*e*) => {

*e*.preventDefault()

        changeStep(currentStep - 1, *e*)

        navigate('/avaliacao/avalie')

    }

    // Objeto com os emojis para fazer a comparação e recuperar do usuário

    const emojiData = {

        unsatisfied: <*BsFillEmojiFrownFill*/>,

        neutral: <*BsFillEmojiNeutralFill*/>,

        satisfied: <*BsFillEmojiSmileFill*/>,

        verySatisfied: <*BsFillEmojiHeartEyesFill*/>

    }

  return (

    <*Home.Content*>

        <*Info* *titulo*="Falta pouco..." *p*="A sua opinião é muito importante" />

        <*Home.FormContainer*>

            <*Steps* *currentStep*={currentStep}/>

            <*Home.inputContainer* *style*={{gap: "5px"}}>

                <*E.resumoContent* *style*={{display: "flex"}}>

                    <h3>Aqui está o resumo da sua avaliação</h3>

                    <p>Satisfação da sessão:  {emojiData[data.review]}</p>

                    <p>Comentário:  {data.comment}</p>

                    <p>Quanto voltaria na próxima sessão:  {data.comeback}</p>

                </*E.resumoContent*>

            </*Home.inputContainer*>

            <*Home.buttonContent*>

                <*Home.buttonAvancar* *onClick*={(*e*) => previousPage(*e*)}>Voltar</*Home.buttonAvancar*>

                <*Home.buttonAvancar* *onClick*={(*e*) => sendInfo(*e*)}>Enviar</*Home.buttonAvancar*>

            </*Home.buttonContent*>

        </*Home.FormContainer*>

    </*Home.Content*>

  )

}

Adm.jsx

// Página de Administrador

\* @decription Página de adm para psicóloga acessar os dados

\* @author Anderson

import React, { useContext, useEffect, useRef } from 'react'

import { Admin, Layout, Resource } from 'react-admin'

import { dataProvider } from './DataProvider'

import { UserList } from '../../Components/UserList/UserList'

import { AppBar } from 'react-admin';

import { styled } from '@mui/material/styles';

import { Dashboard } from '../../Components/Dashboard/Dashboard'

import { Context } from '../../Context/Context'

// Estilizando o AppBar usando Material-UI

const CustomAppBar = styled(AppBar)(({ *theme* }) => ({

  backgroundColor: '#457B9D',

  color: '#FFFFFF',

  boxShadow: 'none',

  padding: '0 20px',

  height: '50px',

  position: 'absolute',

  display: 'flex',

  justifyContent: 'center',

  width: '100%',

}));

// Criando um layout customizado com o AppBar

const CustomLayout = (*props*) => (

  <*Layout* {...props} *appBar*={CustomAppBar} />

);

export const Adm = () => {

  // usando isMounted para ver se o componente foi remontado

  const isMounted = useRef(false)

  const { loginData, getAllFeedBacks } = useContext(Context)

  // UseEffect to keep user logged in

  useEffect(() => {

    if (isMounted.current) {

      if (loginData.name !== "" && loginData.password !== "") {

        getAllFeedBacks()

      }

    } else {

      isMounted.current = true

    }

  }, [])

  return (

    <*Admin* *dashboard*={Dashboard} *layout*={CustomLayout} *dataProvider*={dataProvider} *basename*="/adm" >

      <*Resource* *name*='users' *list*={UserList} />

    </*Admin*>

  )

}

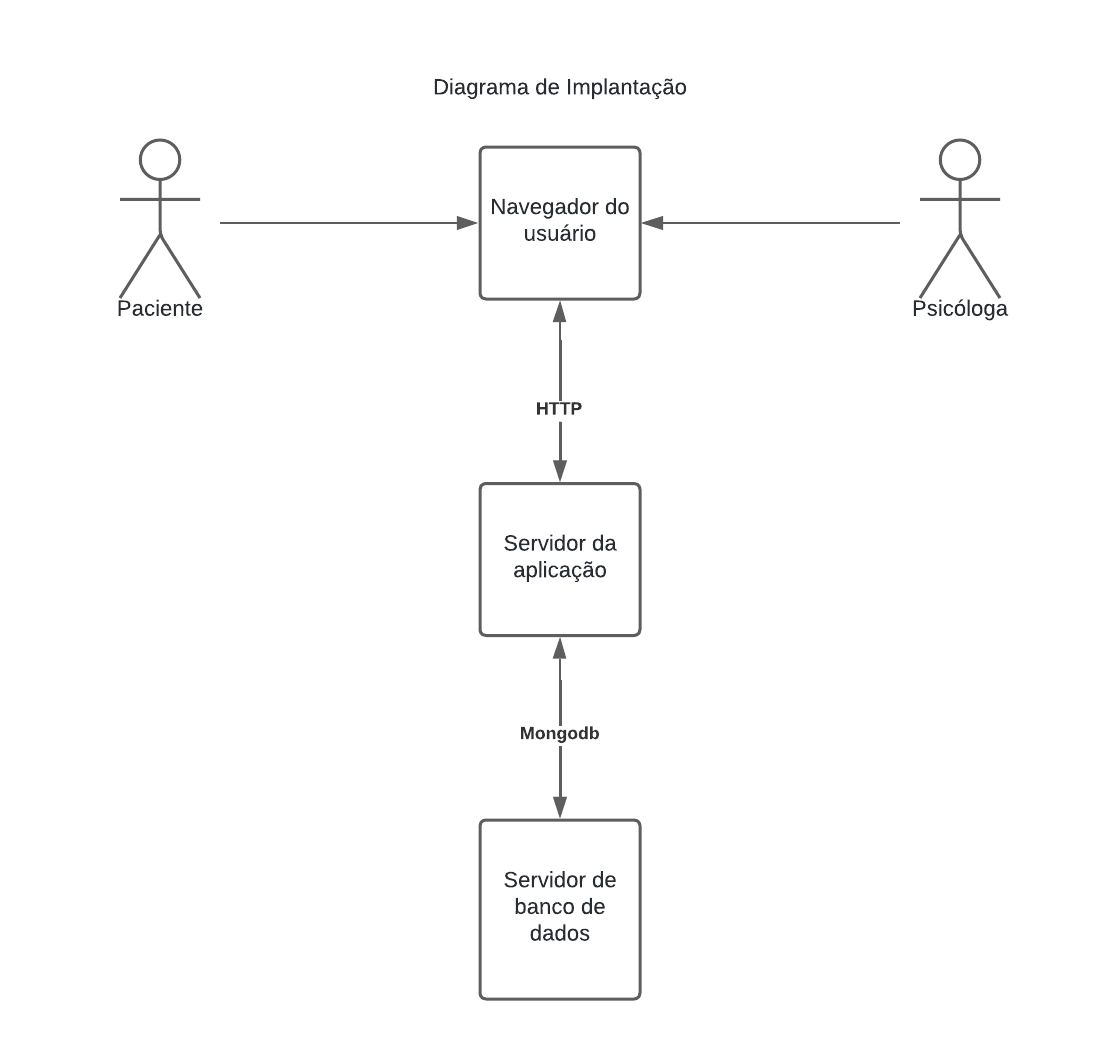
1. Testes
   1. Plano de Testes

|  |  |  |
| --- | --- | --- |
| Nº Do Teste | Descrição do Teste | Resultado Esperado |
| 1 | Tela de login com credenciais válidas | Acesso concedido, redirecionamento para o painel |
| 2 | Teste de login com credenciais inválidas | Mensagem de erro “Invalid credencials” |
| 3 | Teste de envio de feedback preenchendo todos os campos obrigatórios | Feedback adicionado com sucesso |
| 4 | Teste de envio de feedback sem preencher os campos obrigatórios | Mensagem de erro “Campo obrigatório” |
| 5 | Teste de recuperação de senha com e-mail válido | E-mail de recuperação enviado |
| 6 | Teste de recuperação com e-mail inválido | Mensagem de erro “E-mail não encontrado” |
| 7 | Teste de segurança com acesso não autorizado | Acesso negado, usuário redirecionado para tela de login |

* 1. Execução do Plano de Testes

|  |  |  |
| --- | --- | --- |
| Nº do Teste | Resultado Obtido | Comentários |
| 1 | Acesso concedido, redirecionamento para o painel | Teste realizado com sucesso. |
| 2 | Mensagem de erro “Crendenciais inválidas” | Teste realizado com sucesso. |
| 3 | Feedback adicionado com sucesso | Teste realizado com sucesso. |
| 4 | Mensagem de erro “Campo obrigatório” | Teste realizado com sucesso. |
| 5 | E-mail de recuperação enviado | Teste realizado com sucesso. |
| 6 | Mensagem de erro “E-mail não encontrado” | Teste realizado com sucesso. |
| 7 | Acesso negado, usuário redirecionado para a tela de login | Teste realizado com sucesso. |

1. **Implantação**
   1. **Diagrama de Implantação**



**Descrição**

1. Navegador do usuário: Os usuários (pacientes) acessam a aplicação através do navegador, onde podem enviar feedbacks sobre as sessões. Assim como a psicóloga também acessa o navegador para acessar o painel administrativo.
2. Servidor de Aplicação: O Node.js gerencia as requisições da API, processa os dados recebidos do front-end, interage com o banco de dados para armazenar ou recuperar feedbacks.
3. Banco de Dados: MongoDB armazena os feedbacks, informações dos usuários e outros dados relevantes.
   1. **Manual de Implantação**
4. Pré-requisitos

* Hardware e software necessários.
  + Servidor:
    - Processador: Mínimo de 2 núcleos.
    - Memória RAM: 4GB.
    - Armazenamento: 120GB.
  + Máquinas Clientes:
    - Processador: Mínimo de 2 núcleos.
    - Memória RAM: 4GB.
    - Sistema Operacional: Windows 10, macOS, ou Linux Ubuntu 20.04.
  + Software:
    - Node.js (versão 14 ou superior)
    - MongoDB (versão 4 ou superior)
    - NPM (geralmente instalado com Node.js)
* Dependências e configurações do ambiente.
  + Configure variáveis de ambientes necessárias, como PORT para o servidor Node.js.

1. Passo a Passo da instalação

* Passo 1: Preparar o ambiente.
  + Configurar o Servidor:
    - Instale o sistema operacional desejado.
    - Instale o Node.js. Siga as Instruções específicas do sistema operacional.
    - Configure o firewall para permitir conexões nas portas necessárias.
* Passo 2: Instalar os componentes do Sistema
  + Instalação do Servidor Node.js:
    - No servidor, clone o respositório do backend:

git clone <URL\_DO\_REPOSITORIO\_BACKEND>

cd <NOME\_DO\_DIRETORIO>

npm install

* + - Configure o arquivo de ambiente para apontar para o backend.
* Passo 3: Configurar as Conexões de Rede
  + Configurar o Servidor Node.js:
    - Edite o arquivo de configuração para definir a porta do MongoDB.
  + Configurar o cliente React:
    - Verifique o arquivo de configuração do frontend para garantir que está apontando para o endPoint (endereço) do servidor Node.js.
* Passo 4: Testar a instalação:
  + Verificar o MongoDB:
    - Use um cliente MongoDB (como MongoDB Compass) para conectar-se ao banco de dados e verificar se está funcionando.
  + Testar o Servidor Node.js:
    - Inicie o servidor:

Npm run dev

* + - Acesse a URL do servidor em um navegador para verificar se a API está respondendo.
  + Testar o Cliente React:
    - Inicie o cliente:

Npm start

1. Soluções de Problemas Comuns
   1. Erro de Conexão ao MongoDB:
      1. Verifique se o MongoDB está em execução e se a URI está correta.
   2. Servidor Node.js não responde:
      1. Confirme se o serviço do Node.js está em execução e se a porta está correta.
   3. Problemas de Firewall:
      1. Certifique-se de que o firewall não esteja bloqueando as portas necessárias.
2. Referências
   1. Consulte a documentação oficial do Node.js, React, e MongoDB para informações adicionais sobre a instalação.
3. **Manual do Usuário**
   1. **Introdução**

Este manual tem como objetivo fornecer orientações claras e consisas sobre como utilizar o sistema desenvolvido em Node.js, React e MongoDB. A seguir, você encontrará instruções passo a passo para acessar e navegar pelas principais funcionalidades do sistema.

* 1. **Acesso ao Sistema**
     1. **Acessar a Aplicação**

1. Abra o navegador:
   1. Utilize um navegador compatível, como Google Chrome, Firefox ou Safari.
2. Digite a URL:
   1. Insira a URL do sistema no campo de endereços.
      1. **Login no Sistema**
3. Tela de Login:
   1. Na página inicial, você verá um formulário de login.
4. Preencher as Credenciais:
   1. Insira seu nome de usuário e senha.
5. Clicar em “Entrar”:
   1. Após preencher as informações, clique no botão entrar.
      1. **Enviar Feedbacks**
6. Identificação:
   1. Na página inicial, você verá um formulário para colocar nome e e-mail.
   2. Caso não queira se identificar bastar clicar em permanecer anônimo.
7. Avaliação:
   1. Na página de avaliação, você verá opções para avaliar a sessão e um campo para deixar um comentário quaso necessário.
8. Avalie:
   1. Na página avalie, você verá um campo para avaliar de 1 a 5 o quanto você voltaria para uma nova sessão.
9. Envio:
   1. Na página de envio, você verá um breve resumo da sua avaliação, conforme tudo certo basta clicar em enviar.
   2. **Navegação pelo Sistema**
      1. **Menu Principal**

Após o login, você será direcionado ao menu principal, onde pode acessar as seguintes seções:

1. Dashboard:
   1. Visualize um resumo das principais métricas e informações das sessões avaliadas.
   2. Busca de Filtros: Utilize a barra de busca na parte superior da lista de feedbacks para encontrar filtros especificos
   3. Aplique filtros para refinar sua busca como: Todos, Anônimos, Não Anônimos.

**Glossário**