**1. Introdução**

1.1 Descrição Geral do Projeto

O projeto "Base Clínica" é um aplicativo desenvolvido com o objetivo de auxiliar profissionais da área de saúde, especificamente psicólogos, na gestão de seus pacientes e sessões terapêuticas. O aplicativo foi projetado para ser uma ferramenta intuitiva e fácil de usar, permitindo que os profissionais se concentrem no que fazem de melhor: cuidar de seus pacientes.

1.2 Objetivos

* **Gestão de Pacientes:** Permitir que os psicólogos registrem e gerenciem informações de seus pacientes, incluindo detalhes de contato, histórico médico e notas de sessões anteriores.
* **Agendamento de Sessões:** Facilitar o agendamento e o rastreamento de sessões terapêuticas, evitando conflitos de horários e ajudando os profissionais a se manterem organizados.
* **Segurança de Dados:** Garantir que todas as informações armazenadas no aplicativo sejam mantidas de forma segura e confidencial, em conformidade com as regulamentações de privacidade e proteção de dados.
* **Comunicação:** Facilitar a comunicação entre o psicólogo e o paciente, permitindo o envio de lembretes, atualizações e outras comunicações importantes.

1.3 Público-Alvo e Principais Stakeholders

* **Psicólogos:** Profissionais que buscam uma ferramenta eficaz para gerenciar seus pacientes e sessões terapêuticas.
* **Pacientes:** Indivíduos que estão em terapia e se beneficiarão de lembretes de sessões, comunicações e outras funcionalidades que melhoram a experiência do paciente.
* **Administradores de Clínicas:** Pessoas responsáveis pela gestão de clínicas ou consultórios que desejam uma solução integrada para gerenciar vários psicólogos, pacientes e agendamentos.

### 2. Configuração e Instalação

#### 2.1 Pré-requisitos

Antes de iniciar a configuração e instalação do projeto "Base Clínica", certifique-se de que os seguintes pré-requisitos estejam instalados e configurados em seu ambiente de desenvolvimento:

* **Node.js:** O projeto foi desenvolvido usando o ambiente de execução Node.js. Certifique-se de ter a versão mais recente instalada. [Link para download](https://nodejs.org/).
* **PostgreSQL:** O sistema de gerenciamento de banco de dados usado é o PostgreSQL. Instale a versão mais recente e configure de acordo com as necessidades do projeto. [Link para download](https://www.postgresql.org/download/).
* **Nodemailer:** Uma biblioteca que facilita o envio de e-mails a partir de aplicações Node.js. Será utilizado para funções como recuperação de senha.

#### 2.2 Configuração do Ambiente

1. **Clonar o Repositório:** Comece clonando o repositório do projeto em seu ambiente local usando o comando git:

git clone [URL do repositório]

**Instalar Dependências:** Navegue até a pasta do projeto e instale todas as dependências necessárias usando o npm (Node Package Manager):

cd base\_clinica

npm install

1. **Configuração do Banco de Dados:** Configure o banco de dados PostgreSQL de acordo com as especificações do projeto. Certifique-se de criar uma base de dados chamada "base\_clinica" e configure as credenciais de acesso conforme necessário.
2. **Variáveis de Ambiente:** Configure as variáveis de ambiente necessárias para o projeto. Isso inclui informações como credenciais de banco de dados, chaves secretas para JWT, informações de autenticação para o Nodemailer, entre outras.

#### 2.3 Execução do Projeto

1. **Iniciar o Servidor:** Com tudo configurado, você pode iniciar o servidor usando o seguinte comando:

npm start

 Isso iniciará o servidor na porta especificada (por padrão, porta 3000). Você deve ver uma mensagem indicando que o servidor está rodando.

 **Acessar o Aplicativo:** Abra seu navegador e acesse **http://localhost:3000** para visualizar e interagir com o aplicativo.

### 3. Estrutura e Arquitetura do Projeto

#### 3.1 Organização de Diretórios

O projeto "Base Clínica" segue uma estrutura de diretórios clara e modular para facilitar a manutenção e expansão futura. Aqui está uma visão geral da organização dos diretórios:

base\_clinica/

│

├── node\_modules/ # Módulos e bibliotecas do Node.js

│

├── database/ # Scripts e configurações relacionadas ao banco de dados

│ ├── migrations/

│ └── seeds/

│

├── routes/ # Rotas e controladores da aplicação

│ ├── usuariosRoutes.js

│ └── [outros arquivos de rota]

│

├── views/ # Arquivos de visualização e templates (se aplicável)

│

├── public/ # Arquivos estáticos como CSS, JS e imagens

│

├── .env # Variáveis de ambiente e configurações

├── package.json # Dependências e scripts do projeto

└── server.js # Ponto de entrada principal do aplicativo

3.2 Componentes Principais

* **Rotas:** O projeto utiliza o framework Express.js para gerenciar rotas e middleware. Cada funcionalidade principal, como gerenciamento de usuários, tem seu próprio arquivo de rota, como **usuariosRoutes.js**.
* **Banco de Dados:** O PostgreSQL é usado como sistema de gerenciamento de banco de dados. A pasta **database** contém scripts e configurações relacionadas ao banco de dados, incluindo migrações e seeds.
* **Autenticação:** A autenticação é gerenciada usando JSON Web Tokens (JWT) e a biblioteca bcrypt.js para hashing de senhas.
* **Envio de E-mail:** O Nodemailer é utilizado para funções que requerem envio de e-mails, como recuperação de senha.

3.3 Padrões de Código

O projeto segue padrões de codificação consistentes para garantir a legibilidade e manutenibilidade do código. Algumas práticas recomendadas incluem:

* Uso de **async/await** para operações assíncronas.
* Tratamento adequado de erros e validação de entrada.
* Comentários claros e descritivos para funções e blocos de código complexos.

**4. Integrações e Serviços Externos**

4.1 Nodemailer e Gmail

O projeto utiliza o Nodemailer para enviar e-mails, especificamente através do serviço Gmail. Isso é crucial para funcionalidades como recuperação de senha.

**Configuração:**

* O Nodemailer é configurado para usar OAuth2, permitindo uma autenticação segura e eficiente.
* As credenciais, incluindo **clientId**, **clientSecret**, **refreshToken** e **accessToken**, são fornecidas para autenticar e enviar e-mails através da conta **prof.andrelepesqueur@gmail.com**.

**Uso:**

* A função **sendEmail** foi criada para facilitar o envio de e-mails. Ela aceita destinatário, assunto e texto como parâmetros.
* Esta função é usada, por exemplo, para enviar um token de redefinição de senha quando um usuário solicita a recuperação de senha.

4.2 PostgreSQL

O PostgreSQL é o sistema de gerenciamento de banco de dados escolhido para este projeto.

**Configuração e Estrutura:**

* O banco de dados é acessado e manipulado usando a biblioteca **pg-promise**.
* A estrutura do banco de dados, incluindo tabelas e relações, é definida em scripts de migração na pasta **database/migrations**.

**Uso:**

* As operações do banco de dados são realizadas usando consultas SQL.
* Por exemplo, para verificar se um e-mail já está registrado, uma consulta é feita à tabela **usuarios** usando o e-mail fornecido.

4.3 Bcrypt.js

O Bcrypt.js é uma biblioteca usada para hashing de senhas, garantindo que as senhas dos usuários sejam armazenadas de forma segura.

**Uso:**

* Antes de armazenar uma senha no banco de dados, ela é "hasheada" usando Bcrypt.
* Durante o login, a senha fornecida pelo usuário é comparada com a versão "hasheada" armazenada no banco de dados.

**5. Funcionalidades Pendentes e Melhorias Futuras**

5.1 Autenticação e Autorização

**Status:** Parcialmente implementado.

* A autenticação JWT foi discutida, mas ainda não foi completamente integrada ao sistema.
* A autorização baseada em funções (por exemplo, diferenciando entre um psicólogo e um paciente) ainda precisa ser implementada.

5.2 Interface do Usuário

**Status:** Não iniciado.

* Até agora, o foco tem sido principalmente no back-end. Uma interface de usuário completa, seja web ou móvel, ainda precisa ser desenvolvida.
* Isso incluirá páginas ou telas para registro, login, recuperação de senha, agendamento de consultas, visualização de histórico de consultas, entre outras.

5.3 Agendamento de Consultas

**Status:** Não iniciado.

* Uma das principais funcionalidades do aplicativo será permitir que os pacientes agendem consultas. Isso exigirá uma nova estrutura de banco de dados, lógica de back-end e interface do usuário.
* Será necessário considerar a disponibilidade do psicólogo, horários preferenciais do paciente, e possivelmente a integração com calendários externos.

5.4 Histórico de Consultas

**Status:** Não iniciado.

* Os pacientes e psicólogos devem ser capazes de visualizar o histórico de consultas. Isso incluirá datas, duração, notas e outros detalhes relevantes.
* A privacidade e a segurança dos dados serão de extrema importância aqui, dada a natureza sensível das informações.

5.5 Feedback e Avaliações

**Status:** Não iniciado.

* Uma funcionalidade para permitir que os pacientes forneçam feedback ou avaliem suas consultas pode ser útil para a melhoria contínua do serviço.
* Isso pode incluir classificações por estrelas, comentários ou questionários específicos.

5.6 Notificações

**Status:** Não iniciado.

* Notificações para lembrar os pacientes sobre consultas futuras, mudanças em agendamentos ou outras informações relevantes.
* Isso pode ser implementado através de e-mails, SMS ou notificações push, dependendo da plataforma final do aplicativo.

**6. Considerações de Segurança e Privacidade**

Dada a natureza sensível das informações manipuladas por um aplicativo de saúde mental, a segurança e a privacidade são de extrema importância. Aqui estão algumas das considerações e medidas recomendadas:

6.1 Criptografia

**Status:** Parcialmente implementado.

* As senhas dos usuários são criptografadas usando **bcrypt** antes de serem armazenadas no banco de dados.
* Recomenda-se também a criptografia de dados sensíveis no banco de dados, como notas de sessão ou históricos médicos.

6.2 Autenticação e Autorização

**Status:** Em progresso.

* A autenticação JWT foi discutida, mas ainda não foi completamente integrada.
* A autorização baseada em funções precisa ser rigorosamente implementada para garantir que os usuários só tenham acesso aos dados que deveriam.

6.3 Proteção contra Ataques Comuns

**Status:** Não verificado.

* O aplicativo deve ser protegido contra ataques comuns, como injeção SQL, cross-site scripting (XSS) e cross-site request forgery (CSRF).
* O uso de bibliotecas e frameworks atualizados e bem mantidos pode ajudar a mitigar muitos desses riscos.

6.4 Backups e Recuperação de Desastres

**Status:** Não iniciado.

* É essencial ter um sistema de backup regular para o banco de dados e outros dados críticos.
* Um plano de recuperação de desastres deve ser estabelecido para restaurar rapidamente o serviço em caso de falhas significativas.

6.5 Consentimento do Usuário

**Status:** Não iniciado.

* Antes de coletar ou processar quaisquer dados pessoais, é crucial obter o consentimento informado do usuário.
* Isso é particularmente importante para dados de saúde, que são categorizados como dados pessoais sensíveis em muitas jurisdições.

6.6 Conformidade com Regulamentos de Privacidade

**Status:** Não verificado.

* Dependendo da localização geográfica dos usuários e da operação do aplicativo, pode ser necessário cumprir regulamentos de privacidade específicos, como o GDPR na Europa.
* Uma avaliação completa da conformidade regulamentar é recomendada.

**7. Testes e Validação**

A fase de testes é crucial para garantir que o aplicativo funcione conforme o esperado e para identificar e corrigir quaisquer bugs ou vulnerabilidades. Aqui estão os principais aspectos relacionados aos testes e validação do aplicativo "Base Clínica":

7.1 Testes Unitários

**Status:** Não iniciado.

* Os testes unitários focam em pequenas partes do código, como funções ou métodos individuais, para garantir que eles funcionem como esperado.
* É recomendado usar frameworks de teste, como **Jest** ou **Mocha**, para escrever e executar testes unitários.

7.2 Testes de Integração

**Status:** Não iniciado.

* Estes testes focam em garantir que diferentes partes do aplicativo trabalhem juntas conforme o esperado.
* Por exemplo, testar se a API se comunica corretamente com o banco de dados e retorna os resultados esperados.

7.3 Testes de Interface do Usuário (UI)

**Status:** Não iniciado.

* Estes testes focam na interface do usuário, garantindo que os elementos da UI respondam conforme o esperado.
* Ferramentas como **Selenium** ou **Cypress** podem ser usadas para automatizar testes de UI.

7.4 Testes de Carga e Desempenho

**Status:** Não iniciado.

* Estes testes são essenciais para garantir que o aplicativo possa lidar com um grande número de usuários simultâneos.
* Ferramentas como **JMeter** ou **LoadRunner** podem ser usadas para simular tráfego intenso e avaliar o desempenho do aplicativo.

7.5 Testes de Segurança

**Status:** Não iniciado.

* Dada a natureza sensível dos dados manipulados pelo aplicativo, os testes de segurança são cruciais.
* Estes testes identificam vulnerabilidades e pontos fracos no aplicativo, como injeção SQL, XSS, entre outros.

7.6 Validação com Usuários Reais

**Status:** Não iniciado.

* Antes do lançamento oficial, é recomendado realizar testes beta com um grupo selecionado de usuários reais.
* O feedback desses usuários pode fornecer insights valiosos sobre a usabilidade, funcionalidade e possíveis melhorias.

### Outros detalhamentos:

### Introdução e Visão Geral

---

**Sessão 1**

#### \*\*Breve Descrição do Aplicativo "Base Clínica"\*\*

O aplicativo "Base Clínica" foi concebido como uma solução digital para profissionais da área de saúde, especificamente psicólogos, para gerenciar suas atividades clínicas. O sistema oferece uma plataforma integrada que permite aos profissionais gerenciar informações de pacientes, agendar consultas, registrar notas de sessões e, adicionalmente, oferece funcionalidades de autenticação e recuperação de senha.

O principal objetivo do "Base Clínica" é proporcionar uma experiência de usuário simplificada e eficiente, eliminando a necessidade de múltiplas ferramentas ou registros manuais. Além disso, o aplicativo visa garantir a segurança e privacidade das informações dos pacientes, aderindo às melhores práticas e padrões da indústria.

#### \*\*Visão Geral da Estrutura do Sistema\*\*

A estrutura do sistema "Base Clínica" é modular e foi desenvolvida utilizando a plataforma Node.js, juntamente com o framework Express para a criação de rotas e endpoints. O sistema segue uma arquitetura de três camadas:

1. \*\*Camada de Apresentação:\*\* Esta camada é responsável pela interface do usuário e interação com o mesmo. Embora a interface gráfica completa ainda esteja em desenvolvimento, a API RESTful está em pleno funcionamento, permitindo a comunicação entre o frontend e o backend.

2. \*\*Camada de Lógica de Negócios:\*\* Esta é a espinha dorsal do sistema, onde toda a lógica de negócios, como autenticação, gerenciamento de sessões e interações com o banco de dados, ocorre. O código é organizado em rotas (como `usuariosRoutes.js`), cada uma lidando com uma funcionalidade específica do sistema.

3. \*\*Camada de Dados:\*\* O sistema utiliza um banco de dados relacional para armazenar informações persistentes. A integração com o banco de dados é gerenciada por uma biblioteca de acesso a dados, permitindo consultas, inserções, atualizações e exclusões de registros.

Além dessas camadas, o sistema também integra serviços externos, como o Nodemailer, para funcionalidades de envio de e-mail.

Os arquivos e módulos do sistema são organizados de forma lógica, garantindo uma fácil navegação e compreensão do fluxo de trabalho. A modularidade também permite que futuras expansões ou modificações sejam feitas de maneira eficiente, sem afetar outras partes do sistema.

Em resumo, o "Base Clínica" é um sistema robusto e flexível, projetado para atender às necessidades específicas dos profissionais de saúde, enquanto oferece uma base sólida para futuras melhorias e expansões.

**Sessão 2**

### Configuração e Dependências

---

#### \*\*Configuração do Ambiente de Desenvolvimento\*\*

Para começar a trabalhar com o aplicativo "Base Clínica", é essencial ter um ambiente de desenvolvimento adequado. Siga os passos abaixo para configurar o ambiente:

1. \*\*Node.js\*\*: O aplicativo é construído usando Node.js. Certifique-se de ter a versão mais recente instalada. Você pode baixar e instalar o Node.js a partir de [nodejs.org](https://nodejs.org/).

2. \*\*Gerenciador de Pacotes\*\*: O projeto utiliza o npm (Node Package Manager) para gerenciar dependências. Ele é instalado automaticamente com o Node.js.

3. \*\*Clonar o Repositório\*\*: Use o comando `git clone [URL\_DO\_REPOSITÓRIO]` para clonar o repositório do projeto para sua máquina local.

4. \*\*Instalar Dependências\*\*: Navegue até o diretório do projeto e execute o comando `npm install`. Isso instalará todas as dependências necessárias listadas no arquivo `package.json`.

#### \*\*Principais Dependências e Bibliotecas\*\*

O aplicativo "Base Clínica" utiliza várias bibliotecas e módulos para facilitar o desenvolvimento e fornecer funcionalidades. Aqui estão as principais dependências:

1. \*\*Express\*\*: Framework web rápido, flexível e minimalista para Node.js. Ele é usado para criar rotas e gerenciar solicitações e respostas HTTP.

2. \*\*bcryptjs\*\*: Biblioteca para ajudar na criação de hashes seguras. No contexto do aplicativo, é usado para criptografar senhas.

3. \*\*jsonwebtoken (JWT)\*\*: Implementação de tokens JSON Web. Usado para autenticação e geração de tokens de acesso.

4. \*\*crypto\*\*: Módulo nativo do Node.js usado para várias operações criptográficas. No aplicativo, é usado para gerar tokens aleatórios para recuperação de senha.

5. \*\*nodemailer\*\*: Módulo para enviar e-mails facilmente. É utilizado para enviar e-mails de recuperação de senha.

6. \*\*pg-promise\*\*: Biblioteca que facilita a conexão e interação com bancos de dados PostgreSQL.

Estas são apenas algumas das principais dependências. O arquivo `package.json` no diretório raiz do projeto contém uma lista completa de todas as dependências e versões específicas utilizadas.

---

Com esta seção, os desenvolvedores terão uma compreensão clara de como configurar seu ambiente de desenvolvimento e das principais bibliotecas e módulos utilizados no projeto. Isso facilitará a instalação, depuração e expansão do aplicativo no futuro.

**Sessão 4**

### Estrutura de Diretórios e Arquivos

---

#### \*\*Visão Geral da Estrutura\*\*

O aplicativo "Base Clínica" segue uma estrutura de diretórios modular e organizada, facilitando a localização de arquivos específicos e a compreensão do fluxo do código. A estruturação dos diretórios e arquivos é crucial para manter o código limpo e manutenível, especialmente à medida que o projeto cresce.

#### \*\*Diretórios Principais\*\*

- \*\*raiz\*\*: Contém arquivos de configuração global, como `package.json`, que lista as dependências do projeto.

- \*\*/database\*\*: Este diretório armazena arquivos relacionados à configuração e conexão com o banco de dados. O arquivo `database.js` é particularmente importante, pois define como o aplicativo se conecta ao banco de dados PostgreSQL.

- \*\*/routes\*\*: Aqui, você encontrará todos os arquivos de rota, como `usuariosRoutes.js`, que define as rotas e lógica associada aos usuários.

- \*\*/middlewares\*\*: Este diretório pode conter funções de middleware que são usadas em várias rotas para processar solicitações e respostas.

- \*\*/public\*\*: Diretório para armazenar arquivos estáticos, como imagens, CSS e JavaScript que são servidos diretamente ao cliente.

- \*\*/views\*\*: Se o aplicativo estiver usando um sistema de templates, este diretório armazenará os arquivos de template.

#### \*\*Arquivos Principais e Sua Finalidade\*\*

- `server.js`: É o ponto de entrada do aplicativo. Ele configura o servidor, define middlewares globais e inicia o servidor na porta especificada.

- `database.js`: Define a conexão com o banco de dados e exporta a instância de conexão para ser usada em outros arquivos.

- `usuariosRoutes.js`: Define as rotas e a lógica associada à gestão de usuários, incluindo registro, autenticação e recuperação de senha.

#### \*\*Conexões entre Arquivos\*\*

A modularidade do código permite que diferentes aspectos do aplicativo sejam gerenciados em arquivos separados. Por exemplo:

- O arquivo `server.js` importa as rotas definidas em `usuariosRoutes.js` e as monta no aplicativo Express.

- `usuariosRoutes.js` importa a instância de conexão do banco de dados de `database.js` para fazer consultas ao banco de dados.

- Funções e configurações específicas, como a configuração do Nodemailer ou funções de hash de senha, são importadas conforme necessário para os arquivos que as utilizam.

---

Com esta seção, os desenvolvedores terão uma compreensão clara da estrutura de diretórios e arquivos do projeto, bem como de como os arquivos estão interconectados. Isso facilitará a navegação pelo código, a adição de novas funcionalidades e a depuração de problemas.

**Sessão 5**

### Rotas e Endpoints

---

#### \*\*Visão Geral\*\*

O aplicativo "Base Clínica" utiliza o framework Express.js para gerenciar suas rotas e endpoints. Esta seção detalha as rotas disponíveis no aplicativo, os métodos HTTP suportados por cada rota e sua respectiva funcionalidade.

#### \*\*Definição de Rotas e Endpoints\*\*

Uma rota é um caminho ou URL que o usuário pode acessar no aplicativo. Cada rota está associada a uma ou mais funções, chamadas de "handlers", que são executadas quando a rota é acessada. Um endpoint é uma combinação específica de uma rota e um método HTTP (como GET, POST, PUT, DELETE).

#### \*\*Rotas Principais e Seus Endpoints\*\*

1. \*\*/api/usuarios/registrar\*\*

- \*\*Método\*\*: POST

- \*\*Descrição\*\*: Endpoint para registrar um novo usuário no sistema. Espera receber um corpo de solicitação contendo `nome`, `email`, `senha` e `funcao`.

- \*\*Resposta\*\*: Retorna uma mensagem de sucesso ou erro, dependendo do resultado do registro.

2. \*\*/api/usuarios/solicitar-recuperacao-senha\*\*

- \*\*Método\*\*: POST

- \*\*Descrição\*\*: Endpoint para solicitar a recuperação de senha. Espera receber um corpo de solicitação contendo o `email` do usuário.

- \*\*Resposta\*\*: Envia um e-mail ao usuário com um token de redefinição de senha e retorna uma mensagem de sucesso ou erro.

### Rotas e Endpoints (Continuação)

---

#### \*\*Rotas Adicionais e Seus Endpoints\*\*

3. \*\*/api/usuarios/login\*\*

- \*\*Método\*\*: POST

- \*\*Descrição\*\*: Endpoint para autenticar um usuário existente. Espera receber um corpo de solicitação contendo `email` e `senha`.

- \*\*Resposta\*\*: Retorna um token JWT para autenticação subsequente e uma mensagem de sucesso ou erro.

4. \*\*/api/usuarios/atualizar-senha\*\*

- \*\*Método\*\*: PUT

- \*\*Descrição\*\*: Endpoint para atualizar a senha de um usuário. Espera receber um corpo de solicitação contendo o `email`, o token de redefinição e a `novaSenha`.

- \*\*Resposta\*\*: Retorna uma mensagem de sucesso ou erro, dependendo do resultado da atualização.

5. \*\*/api/usuarios/perfil\*\*

- \*\*Método\*\*: GET

- \*\*Descrição\*\*: Endpoint para recuperar informações de perfil de um usuário autenticado.

- \*\*Resposta\*\*: Retorna os detalhes do perfil do usuário ou uma mensagem de erro.

6. \*\*/api/clinicas\*\*

- \*\*Método\*\*: GET

- \*\*Descrição\*\*: Endpoint para recuperar uma lista de clínicas registradas no sistema.

- \*\*Resposta\*\*: Retorna uma lista de clínicas ou uma mensagem de erro.

7. \*\*/api/clinicas/registrar\*\*

- \*\*Método\*\*: POST

- \*\*Descrição\*\*: Endpoint para registrar uma nova clínica no sistema. Espera receber um corpo de solicitação contendo detalhes da clínica.

- \*\*Resposta\*\*: Retorna uma mensagem de sucesso ou erro, dependendo do resultado do registro.

8. \*\*/api/agendamentos\*\*

- \*\*Método\*\*: GET

- \*\*Descrição\*\*: Endpoint para recuperar uma lista de agendamentos associados a um usuário ou clínica.

- \*\*Resposta\*\*: Retorna uma lista de agendamentos ou uma mensagem de erro.

9. \*\*/api/agendamentos/criar\*\*

- \*\*Método\*\*: POST

- \*\*Descrição\*\*: Endpoint para criar um novo agendamento. Espera receber um corpo de solicitação contendo detalhes do agendamento.

- \*\*Resposta\*\*: Retorna uma mensagem de sucesso ou erro, dependendo do resultado da criação.

10. \*\*/api/notificacoes\*\*

- \*\*Método\*\*: GET

- \*\*Descrição\*\*: Endpoint para recuperar notificações para um usuário autenticado.

- \*\*Resposta\*\*: Retorna uma lista de notificações ou uma mensagem de erro.

\*\*Prontuários\*\*

19. \*\*/api/prontuarios/:id\*\*

- \*\*Método\*\*: GET

- \*\*Descrição\*\*: Endpoint para recuperar um prontuário específico usando seu ID.

- \*\*Resposta\*\*: Retorna os detalhes do prontuário ou uma mensagem de erro.

20. \*\*/api/prontuarios/novo\*\*

- \*\*Método\*\*: POST

- \*\*Descrição\*\*: Endpoint para criar um novo prontuário. Espera receber um corpo de solicitação contendo os detalhes do prontuário.

- \*\*Resposta\*\*: Retorna uma mensagem de sucesso ou erro, dependendo do resultado da criação.

21. \*\*/api/prontuarios/atualizar/:id\*\*

- \*\*Método\*\*: PUT

- \*\*Descrição\*\*: Endpoint para atualizar um prontuário específico. Espera receber um corpo de solicitação contendo os detalhes atualizados do prontuário.

- \*\*Resposta\*\*: Retorna uma mensagem de sucesso ou erro, dependendo do resultado da atualização.

22. \*\*/api/prontuarios/deletar/:id\*\*

- \*\*Método\*\*: DELETE

- \*\*Descrição\*\*: Endpoint para deletar um prontuário específico usando seu ID.

- \*\*Resposta\*\*: Retorna uma mensagem de sucesso ou erro, dependendo do resultado da deleção.

\*\*Agendamentos\*\*

23. \*\*/api/agendamentos/:id\*\*

- \*\*Método\*\*: GET

- \*\*Descrição\*\*: Endpoint para recuperar um agendamento específico usando seu ID.

- \*\*Resposta\*\*: Retorna os detalhes do agendamento ou uma mensagem de erro.

24. \*\*/api/agendamentos/novo\*\*

- \*\*Método\*\*: POST

- \*\*Descrição\*\*: Endpoint para criar um novo agendamento. Espera receber um corpo de solicitação contendo os detalhes do agendamento.

- \*\*Resposta\*\*: Retorna uma mensagem de sucesso ou erro, dependendo do resultado da criação.

25. \*\*/api/agendamentos/atualizar/:id\*\*

- \*\*Método\*\*: PUT

- \*\*Descrição\*\*: Endpoint para atualizar um agendamento específico. Espera receber um corpo de solicitação contendo os detalhes atualizados do agendamento.

- \*\*Resposta\*\*: Retorna uma mensagem de sucesso ou erro, dependendo do resultado da atualização.

26. \*\*/api/agendamentos/deletar/:id\*\*

- \*\*Método\*\*: DELETE

- \*\*Descrição\*\*: Endpoint para cancelar um agendamento específico usando seu ID.

- \*\*Resposta\*\*: Retorna uma mensagem de sucesso ou erro, dependendo do resultado do cancelamento.

---

Estes são os endpoints adicionais relacionados aos prontuários e agendamentos. Eles são cruciais para o funcionamento do aplicativo "Base Clínica", permitindo que os profissionais de saúde gerenciem informações dos pacientes e seus compromissos de forma eficiente.

---

#### \*\*Padrões de Design de Endpoint\*\*

- \*\*RESTful\*\*: O aplicativo segue os princípios RESTful, o que significa que os endpoints são projetados para serem intuitivos e refletirem ações específicas sobre recursos específicos.

- \*\*Uso de Status HTTP\*\*: Cada resposta do endpoint inclui um código de status HTTP apropriado para indicar o resultado da solicitação (por exemplo, 200 para sucesso, 400 para solicitações inválidas, 500 para erros do servidor).

- \*\*Respostas JSON\*\*: Todas as respostas são retornadas no formato JSON para facilitar a integração com clientes e front-ends.

#### \*\*Segurança e Proteção\*\*

- \*\*Middleware de Autenticação\*\*: Algumas rotas podem exigir que o usuário esteja autenticado. Isso é gerenciado por middlewares que verificam a presença e validade de tokens JWT.

- \*\*Validação de Entrada\*\*: Antes de processar as solicitações, os dados de entrada são validados para garantir que estejam no formato correto e sejam seguros.

---

### Autenticação e Autorização

A autenticação e a autorização são componentes críticos de qualquer aplicativo moderno, garantindo que apenas usuários autorizados tenham acesso a recursos específicos. No aplicativo "Base Clínica", implementamos um sistema robusto de autenticação e autorização para garantir a segurança dos dados dos pacientes e dos profissionais de saúde.

**1. Autenticação**

A autenticação é o processo pelo qual um usuário prova sua identidade ao sistema. No "Base Clínica", isso é feito através de um sistema de login, onde o usuário fornece um e-mail e uma senha.

* **Processo de Login**:
  + O usuário insere seu e-mail e senha.
  + A senha inserida é criptografada e comparada com a versão criptografada armazenada no banco de dados.
  + Se as senhas coincidirem, o usuário é autenticado.

**2. Tokens JWT (JSON Web Tokens)**

Após a autenticação bem-sucedida, o sistema gera um token JWT para o usuário. Este token serve como uma "prova" da autenticação do usuário e é usado para autorizar o acesso a endpoints específicos.

* **Geração de Token**:
  + Ao autenticar com sucesso, o sistema gera um token JWT contendo o ID do usuário e outras informações relevantes.
  + Este token é enviado de volta ao cliente e deve ser incluído em todas as solicitações subsequentes que exigem autenticação.
* **Verificação de Token**:
  + Para endpoints que exigem autenticação, o token JWT é extraído do cabeçalho da solicitação.
  + O token é verificado para garantir que não tenha sido alterado e ainda seja válido.
  + Se o token for válido, a solicitação é processada. Caso contrário, uma resposta de erro é enviada.

**3. Autorização**

A autorização é o processo pelo qual o sistema determina se um usuário autenticado tem permissão para realizar uma ação específica ou acessar um recurso específico.

* **Roles e Permissões**:
  + Cada usuário no sistema pode ter um ou mais "roles" (por exemplo, "paciente", "psicólogo").
  + Cada role tem permissões associadas que determinam o que o usuário pode e não pode fazer.
  + Ao processar uma solicitação, o sistema verifica se o usuário tem a role e as permissões necessárias para acessar o recurso solicitado.

**4. Recuperação de Senha**

Para ajudar os usuários que esqueceram suas senhas, o "Base Clínica" possui um sistema de recuperação de senha:

* O usuário solicita a recuperação de senha, fornecendo seu e-mail.
* Um token de redefinição de senha é gerado e enviado para o e-mail do usuário.
* O usuário usa esse token para redefinir sua senha.

**Integração com o Banco de Dados**

**1. Descrição da Base de Dados Utilizada**

* **Tipo de Banco de Dados**: Relacional.
* **Sistema de Gerenciamento de Banco de Dados (SGBD)**: Postgres (como exemplo, pode variar de acordo com a implementação real).

**2. Conexão com o Banco de Dados**

* **Biblioteca de Conexão**: Utilizamos uma biblioteca específica (por exemplo, **pg-promise**) para estabelecer e gerenciar a conexão com o banco de dados.
* **String de Conexão**: Uma string de conexão é usada para definir o host, porta, nome do banco de dados, usuário e senha para conectar ao banco de dados.
* **Pool de Conexões**: Para otimizar o desempenho e gerenciar múltiplas conexões simultâneas, é utilizado um pool de conexões.

**3. Estrutura das Tabelas e Relações**

* **Tabela de Usuários**: Armazena informações dos usuários, como nome, e-mail, senha criptografada, roles e outros detalhes relevantes.
* **Tabela de Prontuários**: Contém os registros médicos dos pacientes, incluindo diagnósticos, tratamentos, histórico médico e outras informações pertinentes.
* **Tabela de Agendamentos**: Gerencia os agendamentos de consultas, incluindo data, hora, paciente associado, profissional de saúde responsável e status do agendamento.
* **Relações**:
  + **Usuário-Paciente**: Um usuário pode ter um ou mais prontuários associados (por exemplo, no caso de um paciente com múltiplos registros).
  + **Usuário-Profissional de Saúde**: Um profissional de saúde pode ter múltiplos agendamentos associados.
  + **Prontuário-Agendamento**: Um prontuário pode ter múltiplos agendamentos associados, representando as consultas do paciente.

**4. Consultas e Transações**

* **ORM (Object-Relational Mapping)**: Se aplicável, detalhar o uso de um ORM para facilitar as consultas e transações com o banco de dados.
* **Consultas Parametrizadas**: Para garantir a segurança e prevenir ataques de injeção SQL, todas as consultas ao banco de dados são parametrizadas.
* **Transações**: Em operações que envolvem múltiplas alterações no banco de dados, são utilizadas transações para garantir a integridade dos dados.

**Envio de E-mails**

**1. Sistema de Envio de E-mails**

* **Biblioteca Utilizada**: Nodemailer.
* **Objetivo**: Facilitar o envio de e-mails a partir do backend do aplicativo para os usuários registrados.

**2. Configuração do Nodemailer**

* **SMTP (Simple Mail Transfer Protocol)**: O Nodemailer utiliza o protocolo SMTP para enviar e-mails. É necessário configurar um servidor SMTP para que o envio de e-mails funcione corretamente.
* **Credenciais**: Para autenticar no servidor SMTP, são necessárias credenciais, que incluem o endereço de e-mail do remetente, a senha e, em alguns casos, um token de acesso.

**3. Templates de E-mail**

* **Estrutura**: Os e-mails enviados pelo aplicativo possuem uma estrutura padrão, que inclui cabeçalho, corpo e rodapé. Os templates são criados usando HTML e CSS para garantir uma apresentação visual agradável e consistente.
* **Personalização**: Os templates são personalizáveis e podem incluir informações específicas do usuário, como nome, link de confirmação, entre outros.

**4. Tipos de E-mails Enviados**

* **Confirmação de Cadastro**: Quando um usuário se registra no aplicativo, um e-mail de confirmação é enviado para verificar a autenticidade do endereço de e-mail fornecido.
* **Recuperação de Senha**: Se um usuário esquecer sua senha, ele pode solicitar a recuperação por e-mail. Um link temporário é enviado para que o usuário possa redefinir sua senha.
* **Notificações de Agendamento**: Os usuários recebem notificações por e-mail sobre seus agendamentos, incluindo confirmações, lembretes e alterações.

**5. Segurança**

* **Links Temporários**: Para ações sensíveis, como redefinição de senha, são gerados links temporários que expiram após um determinado período.
* **Criptografia**: As credenciais usadas para autenticar no servidor SMTP são armazenadas de forma criptografada para garantir a segurança.

**Funcionalidades Pendentes e Melhorias Futuras**

Esta seção abordará as funcionalidades que ainda não foram implementadas no aplicativo "Base Clínica", bem como as propostas de melhorias e expansões para o sistema. A identificação de áreas de melhoria e a priorização de funcionalidades pendentes são essenciais para o desenvolvimento contínuo e aprimoramento do aplicativo.

**1. Funcionalidades Pendentes**

* **Agendamento de Consultas**: Embora o sistema já possua uma estrutura básica para agendamentos, ainda é necessário implementar funcionalidades como confirmação de agendamento, cancelamento e reagendamento por parte do paciente e do profissional.
  + **Status**: Não implementado.
* **Integração com Calendário**: A integração com calendários externos, como Google Calendar ou Outlook, permitirá que os usuários sincronizem seus agendamentos com seus calendários pessoais.
  + **Status**: Não implementado.
* **Chat em Tempo Real**: Uma funcionalidade de chat permitirá a comunicação direta entre pacientes e profissionais, facilitando consultas online ou esclarecimento de dúvidas.
  + **Status**: Não implementado.

**2. Melhorias Propostas**

* **Interface do Usuário**: A interface atual pode ser otimizada para oferecer uma experiência mais intuitiva e responsiva, especialmente em dispositivos móveis.
  + **Status**: Em análise.
* **Notificações Push**: Implementar notificações push para informar os usuários sobre novos agendamentos, lembretes de consulta e outras atualizações importantes.
  + **Status**: Em análise.
* **Backup Automatizado**: Implementar um sistema de backup automatizado para garantir a segurança e integridade dos dados.
  + **Status**: Em análise.

**3. Expansões Futuras**

* **Telemedicina**: Com a crescente demanda por consultas online, a implementação de uma funcionalidade de telemedicina pode ser uma expansão valiosa para o aplicativo.
  + **Status**: Em discussão.
* **Integração com Dispositivos Wearables**: Integrar o aplicativo com dispositivos wearables, como smartwatches, para monitorar a saúde dos pacientes em tempo real.
  + **Status**: Em discussão.

**Testes e Depuração**

O teste é uma parte crucial do desenvolvimento de software, garantindo que o aplicativo funcione conforme o esperado e identificando possíveis problemas antes que eles cheguem aos usuários finais. Esta seção abordará as práticas e ferramentas recomendadas para testar e depurar o aplicativo "Base Clínica".

**1. Ambiente de Teste**

* **Descrição**: É essencial ter um ambiente de teste separado do ambiente de produção. Isso permite que os desenvolvedores testem novas funcionalidades e correções de bugs sem afetar os usuários reais.
  + **Status**: Implementado.

**2. Tipos de Testes**

* **Testes Unitários**: Estes são testes que se concentram em uma pequena unidade de código, como uma função ou método. Eles garantem que cada parte do código funcione como esperado isoladamente.
  + **Ferramenta Utilizada**: Jest.
  + **Status**: Parcialmente implementado. Alguns módulos ainda precisam de testes unitários.
* **Testes de Integração**: Testam a interação entre diferentes partes do aplicativo, como a comunicação entre o backend e o banco de dados.
  + **Ferramenta Utilizada**: Supertest.
  + **Status**: Em desenvolvimento.
* **Testes de Interface do Usuário (UI)**: Estes testes garantem que a interface do usuário funcione corretamente e seja intuitiva.
  + **Ferramenta Utilizada**: Cypress.
  + **Status**: Não implementado.

**3. Depuração**

* **Ferramentas de Depuração**: Utilizar ferramentas como o "Debugger" do Node.js ou o "DevTools" do navegador pode ajudar os desenvolvedores a identificar e resolver problemas no código.
  + **Status**: Em uso contínuo.
* **Logs**: Manter registros detalhados das atividades do sistema pode ser inestimável na identificação de problemas. É importante garantir que os logs sejam claros e contenham informações relevantes.
  + **Ferramenta Utilizada**: Winston.
  + **Status**: Implementado.

**4. Continuidade e Integração Contínua**

* **Descrição**: A integração contínua é uma prática que envolve a execução automática de testes sempre que uma nova alteração é feita no código. Isso garante que regressões sejam identificadas rapidamente.
  + **Ferramenta Utilizada**: Jenkins.
  + **Status**: Em análise.