UNIVERSIDADE DE FORTALEZA

VICE REITORIA DE GRADUAÇÃO

CENTRO DE CIÊNCIAS TECNOLÓGICAS

TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

Anna Correia - 2325494

Lais Barbosa - 2322637

Leticia Lucena - 2322682

Lorena Lin - 2322680

Petrus Galvão - 2322674

DOCUMENTAÇÃO TÉCNICA

Dental Connect

FORTALEZA

2025

Anna Correia - 2325494
Lais Barbosa - 2322637
Leticia Lucena - 2322682
Lorena Lin - 2322680
Petrus Galvão - 2322674

DOCUMENTAÇÃO TÉCNICA

Dental Connect

Este documento contém a documentação técnica do Aplicativo Dental Connect de Conexão entre Pacientes e Dentistas desenvolvido na componente curricular N393 - Projeto Aplicado Multiplataforma como requisito para obtenção de nota.

Supervisor: Prof. Bruno Lopes, Me

FORTALEZA

2025

SUMÁRIO

1. INTRODUÇÃO

1.1. CONTEXTO E JUSTIFICATIVA

O setor de serviços de saúde, especialmente o odontológico, enfrenta um desafio significativo quando se trata de emergências fora do horário comercial. Pacientes que sofrem com dores agudas, fraturas dentárias ou outros problemas urgentes após o expediente comercial, nos finais de semana ou feriados, muitas vezes encontram dificuldades para localizar um dentista disponível.

O público-alvo principal inclui pessoas que buscam atendimento imediato, mas também abrange dentistas que desejam expandir sua atuação para além do horário tradicional, gerando uma fonte de renda extra e oferecendo um serviço essencial à comunidade. A falta de uma plataforma centralizada para essa conexão resulta em pacientes sofrendo desnecessariamente e profissionais perdendo a oportunidade de atender a uma demanda real e constante.

O aplicativo Dental Connect surge como a solução para essa problemática. Ele existe para preencher essa lacuna, proporcionando uma ponte direta e eficiente entre pacientes em situações de emergência e dentistas dispostos a atendê-los, garantindo que o cuidado odontológico de qualidade esteja acessível a qualquer momento, e não apenas de segunda a sexta, das 9h às 18h.

1.2. OBJETIVOS

O objetivo deste projeto é desenvolver um aplicativo móvel para conectar pacientes e dentistas, facilitando a busca e agendamento de consultas, com foco principal em atendimentos de emergência fora do horário comercial.

Dito isso, os objetivos específicos do projeto são:

- Criar um fluxo de cadastro e autenticação para pacientes e dentistas, garantindo acesso seguro e personalizado.
- Implementar um sistema de busca avançada que permita aos pacientes encontrar dentistas disponíveis de acordo com a geolocalização, especialidade e tipo de tratamento.

- Desenvolver módulos de gerenciamento de consultas para que pacientes possam agendar, visualizar e gerenciar suas consultas de forma eficiente.
- Configurar as páginas de conteúdo para fornecer informações detalhadas sobre os serviços e procedimentos odontológicos oferecidos.
- Estruturar a navegação principal do aplicativo de forma intuitiva, permitindo que os usuários acessem as funcionalidades de forma rápida e simples.
- Integrar um canal de suporte ao usuário para resolver dúvidas e receber feedback.

1.3. ESCOPO E DELIMITAÇÃO

- **Escopo:**
    - **Cadastro e Autenticação:** O sistema permitirá o cadastro e login seguro para dois tipos de usuários: pacientes e dentistas.
    - **Busca Avançada de Profissionais:** Os pacientes poderão buscar dentistas disponíveis usando filtros de geolocalização, especialidade e tipo de tratamento.
    - **Foco em Emergências:** A funcionalidade principal é conectar pacientes a dentistas para atendimentos de emergência fora do horário comercial.
    - **Gerenciamento de Consultas (Paciente):** Os pacientes terão um módulo para agendar, visualizar e gerenciar suas consultas.
    - **Conteúdo Informativo:** O aplicativo terá páginas com informações sobre serviços e procedimentos odontológicos.
    - **Navegação Intuitiva:** A estrutura do aplicativo permitirá acesso rápido às funcionalidades.
    - **Canal de Suporte:** Haverá um canal integrado para que usuários tirem dúvidas e enviem feedback.
- **Delimitação (Fora do Escopo):**
    - **Módulo Financeiro:** O sistema não processará pagamentos de consultas, nem fará gestão de cobranças ou emissão de notas fiscais. (Seus objetivos mencionam agendamento e conexão, mas não pagamento).
    - **Prontuários Eletrônicos:** O aplicativo não armazenará o histórico médico detalhado (prontuários) dos pacientes nem emitirá receitas

médicas. (O foco é a "conexão" e "agendamento", não a gestão clínica).

- **Gestão de Agenda para Dentistas:** O sistema não funcionará como um software completo de gestão de clínica para o dentista (ex: bloqueio de agenda pessoal, controle de estoque, etc.). (O objetivo foca no gerenciamento de consultas pelo paciente).

2. ENGENHARIA DE REQUISITOS

2.1. REQUISITOS FUNCIONAIS (RFs)

Instrução: Liste as ações e funcionalidades que o aplicativo deve permitir ao usuário executar. Pense no contexto de uso mobile: o que um vendedor ou gerente de pátio precisaria fazer com o app em mãos? Dê ênfase a interações que se beneficiam da mobilidade e do hardware do dispositivo (câmera, por exemplo).

ID	Nome do Requisito	Descrição
RF01	Autenticação de Usuário	O aplicativo deve permitir que um usuário se autentique com seu e-mail e senha. Após o primeiro login bem-sucedido, o app deve manter o usuário logado para acessos futuros, mesmo offline.
RF02	Visualização do Catálogo	O app deve exibir o catálogo de lajes de pedra em uma lista vertical rolável, com uma imagem em miniatura, o tipo do material e o status de cada peça. Os dados devem vir do cache local para agilizar o carregamento.
RF03	Busca e Filtragem de Lajes	O usuário deve poder buscar lajes por nome e filtrar a lista por status ("Disponível", "Reservado", etc.) para encontrar rapidamente o que precisa.
RF04	Detalhamento de Laje	Ao tocar em um item da lista, o app deve navegar para uma tela de detalhes, exibindo a foto em alta resolução, dimensões, lote, fornecedor e demais informações da laje.
RF05	Adicionar Foto via Câmera	Na tela de detalhes de uma laje, o usuário deve ter a opção de acionar a câmera do dispositivo, capturar uma nova foto e enviá-la para ser associada àquele item no sistema.

2.2. REQUISITOS NÃO FUNCIONAIS (RNFs)

- **Desempenho:**
    - RNF01: O tempo de inicialização do aplicativo (cold start) não deve exceder 3 segundos, incluindo a exibição da tela de splash configurada para 2 segundos.
    - RNF02: A navegação entre telas deve ser fluida, utilizando React Navigation com animações nativas (useNativeDriver:

true) para garantir mínimo de 60 frames por segundo durante transições.
    - RNF03: O aplicativo deve otimizar o consumo de dados móveis, utilizando estados de loading durante requisições HTTP e evitando requisições desnecessárias ao backend.
    - RNF04: As requisições à API devem ter timeout configurado e tratamento de erros adequado para evitar travamentos durante carregamento de dados de clínicas, dentistas e consultas.
- **Usabilidade:**
    - RNF05: A interface deve seguir padrões de design mobile responsivo, utilizando componentes nativos do React Native e bibliotecas como React Native Vector Icons para garantir consistência visual.
    - RNF06: O aplicativo deve fornecer feedback visual imediato ao usuário através de estados de loading, mensagens de erro e animações de feedback durante interações (como seleção de cidade e busca de dentistas).
    - RNF07: A navegação deve ser intuitiva, utilizando Stack Navigator do React Navigation para controle do fluxo de telas, garantindo experiência de usuário familiar e previsível.
- **Compatibilidade:**
    - RNF08: O aplicativo deve ser totalmente funcional em dispositivos Android e iOS, utilizando React Native 0.81.4 e Expo SDK 54, garantindo compatibilidade multiplataforma.
    - RNF09: O aplicativo deve suportar orientação portrait, conforme configurado no app.json, garantindo experiência consistente em diferentes tamanhos de tela.
- **Segurança:**
    - RNF10: O token de autenticação JWT do usuário deve ser gerenciado de forma segura através do contexto de autenticação, sendo transmitido via header Authorization em todas as requisições protegidas à API.
    - RNF11: As senhas dos usuários devem ser criptografadas utilizando bcryptjs no backend antes de serem armazenadas no banco de dados Supabase.
    - RNF12: Todas as requisições HTTP devem utilizar HTTPS quando em produção, garantindo comunicação segura entre o aplicativo e o backend.
    - RNF13: O aplicativo deve validar tokens de autenticação em rotas protegidas, retornando erro 403 quando o acesso não for autorizado, conforme implementado no middleware de autenticação.

- **Confiabilidade:**
  - RNF14: O aplicativo deve tratar adequadamente erros de rede, exibindo mensagens claras ao usuário quando não for possível conectar ao servidor backend.
  - RNF15: O aplicativo deve manter a sessão do usuário durante o uso, utilizando React Context API para gerenciamento de estado global de autenticação.
- **Manutenibilidade:**
  - RNF16: O código deve seguir padrões de organização modular, separando responsabilidades entre controllers, services, repositories e entities no backend, facilitando manutenção e evolução do sistema.

3. PROJETO E ARQUITETURA DO SOFTWARE

3.1. ARQUITETURA GERAL

Instrução: Aqui você deve apresentar a arquitetura interna do seu aplicativo. Em vez da arquitetura do servidor, descreva o padrão de design que você usou para organizar o código do app (ex: MVVM, MVI, Clean Architecture). O diagrama deve ilustrar as camadas do aplicativo e como elas interagem entre si.

Exemplo de Texto: O aplicativo RochaForte foi desenvolvido seguindo a arquitetura **MVVM (Model-View-ViewModel)**, recomendada oficialmente pelo Google para o desenvolvimento Android. Esta arquitetura promove a separação de responsabilidades, facilitando a testabilidade e a manutenção do código.

As camadas são divididas da seguinte forma:

1. **View (Apresentação):** Composta por Activities/Fragments e construída com Jetpack Compose. É responsável por exibir os dados na tela e capturar as interações do usuário, delegando toda a lógica para a ViewModel.
2. **ViewModel:** Contém a lógica de apresentação e gerencia o estado da UI. Ela sobrevive a mudanças de configuração (como a rotação da tela) e expõe os dados para a View através de `StateFlows`. A ViewModel não conhece a View, apenas notifica sobre mudanças de estado.
3. **Model (Dados):** Composta pela camada de **Repositório** (Repository), que é a única fonte da verdade para os dados. O Repositório decide de onde buscar os dados: de uma fonte remota (**Remote Data Source**, que consome a API REST) ou de uma fonte local (**Local Data Source**, que acessa o banco de dados Room para dados em cache).

[INSERIR IMAGEM DO DIAGRAMA DA ARQUITETURA MVVM AQUI]

3.2. PROJETO DE DADOS: INTEGRAÇÃO COM API E PERSISTÊNCIA LOCAL

Instrução: Nesta seção, explique como seu aplicativo gerencia os dados. A fonte primária de informações é a API REST, mas um aplicativo moderno quase sempre precisa de uma estratégia de persistência local (um banco de dados no dispositivo) para funcionar como um cache. Descreva como essas duas fontes de dados trabalham juntas. Explique sua estratégia: o app busca os dados da API e os salva localmente para depois exibi-los? Isso permite que o app funcione offline e melhora drasticamente a performance, pois nem sempre é necessário buscar dados na internet.

Exemplo de Texto:

A estratégia de dados do aplicativo RochaForte é projetada para performance e resiliência, utilizando a API REST como a fonte da verdade e um banco de dados local como um cache otimizado. A interação entre essas duas fontes é orquestrada pela camada de Repositório, seguindo o padrão "Single Source of Truth" (Fonte Única da Verdade) no cliente.

O fluxo de dados funciona da seguinte maneira:

1. Requisição de Dados: A ViewModel solicita dados (ex: a lista de lajes) ao Repositório.
2. Consulta à API: O Repositório sempre busca a versão mais recente dos dados na API REST através da nossa camada de rede (Retrofit).
3. Persistência no Cache: Ao receber a resposta da API, o Repositório imediatamente salva os dados recebidos no banco de dados local (Room), sobrescrevendo os dados antigos daquela consulta.
4. Exibição a Partir do Local: A ViewModel observa os dados diretamente do banco de dados local. Assim que os novos dados são salvos, a interface é atualizada automaticamente de forma reativa.

Essa abordagem garante que:

- Performance: A UI é sempre populada a partir do banco de dados local, que é extremamente rápido.
- Suporte Offline: Se o dispositivo estiver sem conexão, o Repositório falha ao buscar na API, mas a UI continua exibindo os últimos dados que foram salvos no cache local, permitindo que o aplicativo continue funcional para consulta.

A tecnologia escolhida para a persistência local foi o Room Persistence Library. A principal entidade armazenada em cache é a `LajeCacheEntity`.

Dicionário de Dados (Tabela `Lajes_Pedra`):

Nome do campo	Tipo de dados	Chave (PK/FK)	Nulo?	Descrição
id	UUID	PK	Não	UUID da laje, vindo da API.
tipo_material	VARCHAR(100)		Não	Ex: "Mármore Carrara", "Granito Preto Absoluto"
dimensoes_cm	JSONB		Não	Objeto JSON com altura, largura, espessura
url_foto_thumb	VARCHAR(255)		Sim	URL de uma versão em miniatura da foto.
status	VARCHAR(20)		Não	"Disponível", "Reservado", "Vendido"
last_updated	TIMESTAMPZ		Não	Timestamp de quando o dado foi Exportar para as Planilhas

3.3. CONSUMO DA API E FLUXO DE NAVEGAÇÃO

instrução: Esta seção tem dois objetivos. Primeiro, referenciar a documentação oficial da API que o seu aplicativo consome. Segundo, e mais importante, apresentar um **Diagrama de Fluxo de Navegação** que mostre as telas do seu aplicativo e como o usuário transita entre elas.

Exemplo de Texto:

O aplicativo consome a API REST do sistema RochaForte, cuja documentação completa, interativa e oficial, desenvolvida com OpenAPI, pode ser acessada através do seguinte link:

- **URL da Documentação da API:** `https://myprofile.github.io/rochaforte-api-docs/`

O fluxo de navegação do usuário dentro do aplicativo foi projetado para ser simples e intuitivo, conforme ilustrado no diagrama a seguir:

Figura 5: Diagrama de Fluxo de Navegação do Aplicativo (Legenda: O fluxo principal inicia na tela de Login. Após o sucesso, o usuário é direcionado para a Home (lista de lajes), de onde pode navegar para a tela de Detalhes de uma laje específica ou para a tela de Cadastro de uma nova laje.)

`[INSERIR IMAGEM DO DIAGRAMA DE FLUXO DE NAVEGAÇÃO AQUI (SPLASH -> LOGIN -> HOME -> DETALHES)]`

4. TECNOLOGIAS E FERRAMENTAS

4.1. STACK DE TECNOLOGIAS

- **Linguagem:** JavaScript (ES6+), escolhida por sua versatilidade e ampla adoção no desenvolvimento mobile com React Native, oferecendo sintaxe moderna e recursos como arrow functions, destructuring e async/await.
- **Framework Mobile:** React Native, selecionado por permitir desenvolvimento multiplataforma (iOS e Android) com uma única base de código, oferecendo performance nativa e acesso às APIs do dispositivo.
- **Plataforma de Desenvolvimento:** Expo, utilizada para simplificar o processo de desenvolvimento, build e deploy da aplicação React Native, fornecendo ferramentas integradas e serviços de desenvolvimento.
- **Backend:** Node.js com Express.js, escolhido por sua eficiência em I/O assíncrono e vasto ecossistema de pacotes npm, permitindo desenvolvimento rápido de APIs REST.
- **Banco de Dados:** Supabase (PostgreSQL), selecionado como Backend-as-a-Service por oferecer autenticação integrada, banco de dados relacional, storage de arquivos e APIs REST automáticas.
- **Autenticação:** Supabase Auth integrado com JWT (JSON Web Tokens) para gerenciamento seguro de sessões de usuário e controle de acesso.
- **Navegação:** React Navigation, biblioteca oficial para implementação de navegação entre telas, utilizando Stack Navigator para controle do fluxo da aplicação.
- **Gerenciamento de Estado:** React Context API, escolhido para gerenciamento de estado global da aplicação, especialmente para dados de autenticação e informações compartilhadas entre componentes.
- **Persistência Local:** AsyncStorage, utilizada para armazenamento local de dados do usuário e cache offline da aplicação.
- **UI Components:** React Native Vector Icons para ícones customizados e componentes nativos do React Native para construção da interface.
- **Comunicação HTTP:** Fetch API nativo do JavaScript para realizar chamadas HTTP à API REST, com tratamento de promises e async/await.

- **Validação e Segurança:** bcryptjs para hash de senhas e jsonwebtoken para geração e validação de tokens de autenticação.

4.2. FERRAMENTAS DE DESENVOLVIMENTO

- **IDE:** Visual Studio Code foi utilizado como ambiente de desenvolvimento principal, aproveitando suas extensões para React Native, JavaScript/JSX, Git e integração com Expo. A IDE oferece suporte completo para desenvolvimento mobile com debugging integrado.
- **Controle de Versão:** Git foi adotado para controle de versão, com repositório hospedado no GitHub. O projeto utiliza branches separadas para diferentes funcionalidades, incluindo branches específicas como "consulta-por-clinica" para desenvolvimento de features isoladas.
- **Framework Mobile:** Expo CLI foi utilizado para inicialização, desenvolvimento, build e execução do projeto React Native, proporcionando ferramentas integradas para desenvolvimento e deploy.
- **Backend:** Node.js com Express.js foi configurado para desenvolvimento da API REST, utilizando middleware para CORS, parsing de JSON e roteamento de endpoints.
- **Banco de Dados:** Supabase Dashboard utilizado para gerenciamento do banco de dados PostgreSQL, configuração de políticas de segurança e monitoramento de dados.
- **Teste de API:** Durante o desenvolvimento, ferramentas como curl via terminal e testes diretos foram utilizados para validar os endpoints da API e verificar a comunicação entre frontend e backend.
- **Gerenciamento de Dependências:** npm foi utilizado para instalação e gerenciamento de pacotes tanto no frontend quanto no backend, com package.json para controle de versões.
- **Build e Deploy:** Expo Build Service foi utilizado para compilação da aplicação para diferentes plataformas, facilitando o processo de distribuição.

5. IMPLEMENTAÇÃO E RESULTADOS

5.1. TELAS DO SISTEMA

Instrução: Esta é a vitrine do seu projeto. Insira imagens (screenshots) das principais telas da sua aplicação. Cada imagem deve ter uma legenda curta e clara explicando sua finalidade. Dê preferência a telas que demonstrem as funcionalidades mais importantes (RFs) que você listou na seção 2.

Exemplo de Texto:

Figura 1: Tela de Login (Legenda: A tela de login é o ponto de entrada do sistema, garantindo o acesso seguro através de autenticação por e-mail e senha.)

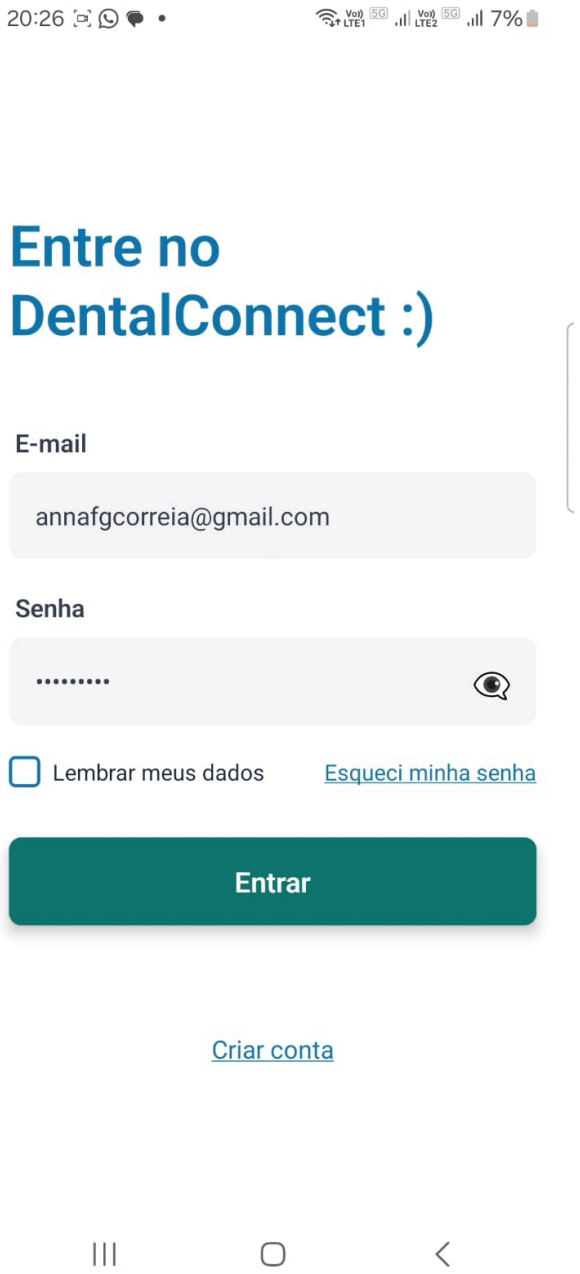

Figura 2: Dashboard Principal com Visão Geral do Estoque (Legenda: Após o login, o administrador tem acesso a um dashboard com métricas rápidas sobre o inventário, como o número de peças disponíveis, reservadas e vendidas.)

[INSERIR SCREENSHOT DO DASHBOARD AQUI]

Figura 3: Tela de Cadastro de Laje de Pedra (Legenda: Formulário detalhado para o cadastro de uma nova peça no inventário, permitindo o upload de foto e a inserção de todas as especificações técnicas.)

[INSERIR SCREENSHOT DO FORMULÁRIO DE CADASTRO AQUI]

6. AMBIENTE E GUIA DE GERAÇÃO (BUILD)

Para compilar o código-fonte e gerar o arquivo APK da plataforma DentalConnect, é necessário configurar o ambiente conforme as especificações indicadas abaixo:

6.1. REQUISITOS DO AMBIENTE

- **IDE:** Visual Studio Code (versão mais recente recomendada)
- **Node.js:** Versão 18.x ou superior (LTS recomendada)
- **npm:** Versão 9.x ou superior (incluído com Node.js)
- **Expo CLI:** Versão 6.x ou superior
- **Git:** Versão 2.x ou superior para controle de versão
- **Sistema Operacional:** Windows 10/11, macOS ou Linux

Também é necessário garantir que as principais dependências e bibliotecas do projeto estejam instaladas, seguindo o processo descrito na seção 6.2.

A versão Mobile do DentalConnect está sendo desenvolvida com React Native 0.81.4 e React 19.1.0, com Expo SDK 54. Entre as bibliotecas principais, estão:

- **react-navigation/native** e **react-navigation/stack**, para controle de navegação entre telas;
- **react-native-async-storage/async-storage**, para armazenamento local de dados;
- **supabase/supabase-js**, para integração com o backend Supabase (autenticação, banco de dados e APIs).
- **axios**, cliente HTTP para consumo de APIs.
- **react-native-calendars**, componente de calendário e agendamentos.
- **expo-linear-gradient**, **expo-status-bar** e **expo/vector-icons**, recursos visuais e ícones nativos do Expo.

6.2. PROCESSO DE GERAÇÃO DE APLICATIVO

Exemplo de Texto:

1. Clone o repositório do projeto front-end: `git clone https://github.com/linlorena/dentalconnect-mobile.git`
2. Clone o repositório do projeto back-end: `git clone https://github.com/laissilva04/dentalconnect-backend.git`
3. Configure o ambiente do front-end: `cd ../dentalconnect-mobile/frontend`
   `npm install`
   `npx expo start`

4. Inicie o servidor backend: `cd ../../dentalconnect-backend`
   `npm start`
   OU
   `npx nodemon index.js`
5. Teste o funcionamento do servidor backend na porta definida.
   `localhost:3000`
6. Gere a versão instalável do APK:
   `npx expo prebuild`
   `npx expo build:android`
   OU
   `npx eas build -p android --profile preview`

6.3. ACESSO À APLICAÇÃO IMPLANTADA

Instrução: Forneça um link para o download direto do arquivo .apk ou para uma plataforma de testes (como Firebase App Distribution) onde a banca possa instalar o aplicativo.

Exemplo de Texto:

- **Link para Download do APK:** [Link do Google Drive/Dropbox para o arquivo .apk]
- **Credenciais de Acesso (Perfil de Vendedor):**
  - **Usuário:** `vendedor@teste.com`
  - **Senha:** `vendedor123`

7. CONCLUSÃO

7.1. TRABALHOS FUTUROS

Instrução: Nenhum projeto está 100% completo. Liste aqui as melhorias e novas funcionalidades que você gostaria de implementar no futuro. Pense em como o aplicativo poderia se tornar ainda mais útil para o usuário. Isso demonstra visão de produto e consciência das limitações do trabalho atual.

Exemplo de Texto:

Com a base sólida do aplicativo estabelecida, identificamos diversas oportunidades para evoluir e agregar ainda mais valor ao sistema RochaForte no futuro. As principais propostas são:

- **Funcionalidades Offline Avançadas:** Expandir a capacidade offline para permitir não apenas a consulta, mas também a **criação e edição de pedidos** sem conexão com a internet. Os dados seriam sincronizados automaticamente com o servidor assim que uma conexão fosse restabelecida.
- **Identificação de Lajes por QR Code:** Implementar uma funcionalidade que utilize a câmera do dispositivo para ler um QR Code fixado em cada laje de pedra. Isso permitiria ao vendedor ou gerente de estoque acessar instantaneamente os detalhes da peça, eliminando a necessidade de busca manual e agilizando o processo de inventário.
- **Notificações Push em Tempo Real:** Desenvolver um sistema de notificações push para alertar os vendedores sobre eventos importantes, como a confirmação de um pedido, a chegada de um novo lote de material ou uma alteração no status de uma laje que ele esteja monitorando.
- **Otimização para Tablets e Modo Paisagem:** Adaptar a interface do usuário para oferecer uma experiência otimizada em telas maiores, como as de tablets, que são frequentemente utilizados em balcões de vendas. Isso incluiria layouts de duas colunas e melhor aproveitamento do espaço horizontal.
- **Versão para a Plataforma iOS:** Iniciar o desenvolvimento da versão do aplicativo para iOS, a fim de atender a todos os potenciais usuários da empresa, independentemente do sistema operacional de seus dispositivos móveis.

7.2. LIÇÕES APRENDIDAS

Instrução: Faça uma reflexão sincera sobre a jornada de desenvolvimento do aplicativo. Quais foram os maiores desafios técnicos que a equipe enfrentou no universo mobile? Quais foram as dificuldades de integrar uma API externa? Como foi o trabalho em equipe? O que vocês fariam de diferente hoje? Esta seção valoriza o processo de aprendizado tanto quanto o resultado final.

Exemplo de Texto:

O desenvolvimento do aplicativo RochaForte foi uma experiência de aprendizado imensa, que nos levou muito além da simples escrita de código. Nossas principais lições aprendidas podem ser divididas em três áreas:

1. **Desafios Técnicos de Arquitetura:** A maior dificuldade técnica que enfrentamos foi, sem dúvida, o gerenciamento de estado e a orquestração de operações assíncronas. Inicialmente, tínhamos dificuldade em manter a interface consistente enquanto os dados eram buscados da API e salvos no banco de dados local. Compreender e aplicar corretamente a arquitetura MVVM, utilizando `StateFlows` para expor o estado da UI a partir da `ViewModel`, foi um divisor de águas. Isso nos ensinou que uma arquitetura bem definida não é opcional, mas sim essencial para criar um aplicativo robusto e manutenível.
2. **A Importância da Persistência Local:** No início, subestimamos a complexidade de oferecer um suporte offline funcional. Implementar o padrão de Repositório como a "Fonte Única da Verdade" que abstrai a origem dos dados (API ou cache local) foi o nosso maior "Aha! Moment". Aprendemos na prática que um aplicativo moderno não apenas consome uma API, mas gerencia dados de forma inteligente para ser rápido e resiliente, melhorando drasticamente a experiência do usuário.
3. **Comunicação entre Equipes (Frontend/Backend):** A integração com a API, desenvolvida pela equipe de Web, foi um grande exercício de comunicação. Depender da documentação OpenAPI foi fundamental e nos ensinou o valor de ter um "contrato" claro entre o cliente e o servidor. Tivemos momentos em que precisávamos de um campo extra ou de um formato de dado diferente, e a negociação com a outra equipe para evoluir a API foi um aprendizado valioso sobre o desenvolvimento colaborativo no mundo real.

A principal lição que levamos deste projeto é que a qualidade de um aplicativo mobile não está apenas em sua aparência, mas em sua arquitetura resiliente e na forma como ele lida com as incertezas do mundo real, como falhas de rede.