Documentação Técnica - iStore

1. Introdução

Nome do Aplicativo:

iStore

Desenvolvedores:

- **Thiago de Souza Rodrigues**
- **Lucas Santana Rossato**

Objetivo:

O **iStore** é um aplicativo mobile desenvolvido em **React Native** com o objetivo de facilitar o controle de estoque de uma loja especializada na venda de produtos Apple. O app permite aos funcionários visualizar, adicionar, editar e excluir produtos, como **iPhones**, **MacBooks**, **iPads** e **iPods**, diretamente pelo celular. Essa solução simplifica o gerenciamento do inventário e otimiza a produtividade da equipe.

Justificativa:

Com a popularização dos dispositivos móveis, o celular se tornou o principal meio de acesso à tecnologia no mundo. Muitos jovens têm mais familiaridade com smartphones do que com desktops, o que reforça a necessidade de ferramentas acessíveis e fáceis de usar em dispositivos móveis.

Além disso, as lojas que comercializam produtos Apple frequentemente lidam com um estoque diversificado e de alto valor. Por isso, ferramentas eficientes para o controle de produtos são indispensáveis para garantir precisão e agilidade nas operações.

O **iStore** atende essa demanda ao substituir sistemas tradicionais de desktop por um aplicativo prático e eficiente. Ele oferece maior agilidade para os funcionários no dia a dia, otimizando o fluxo de trabalho da loja e garantindo um atendimento mais rápido e organizado.

Público-Alvo:

O aplicativo foi desenvolvido pensando em pequenas e médias empresas que atuam no mercado de produtos Apple. Ele é ideal para lojistas que desejam modernizar seus

processos de gestão de estoque sem a necessidade de treinamento técnico avançado ou investimento em equipamentos caros.

Resumo do Projeto:

O **iStore** é resultado de um trabalho acadêmico desenvolvido em parceria entre **Thiago de Souza Rodrigues** e **Lucas Santana Rossato** para a disciplina de **Desenvolvimento Mobile I**. Durante o projeto, foram explorados conceitos importantes de desenvolvimento mobile, como:

- Componentes reutilizáveis em **React Native**;
- Consumo de APIs RESTful com operações **GET**, **POST**, **PUT** e **DELETE**;
- Navegação entre telas utilizando **React Navigation**;
- Utilização do **json-server** para simular o backend.

O aplicativo foi testado em um ambiente Android e iOS e tem como principais pilares a simplicidade de uso e a eficiência no controle de estoque.

2. Tecnologias Utilizadas

Plataforma de Desenvolvimento

- **Expo**: Utilizado para facilitar o desenvolvimento, compilação e teste do aplicativo. O Expo oferece uma estrutura robusta para projetos React Native, permitindo uma configuração simplificada e suporte nativo a diversas bibliotecas.
- **Visual Studio Code**: Ambiente de desenvolvimento integrado (IDE) usado para codificar o projeto, com extensões que auxiliam no desenvolvimento em React Native.
- **Android Studio**: Ferramenta utilizada para testar o aplicativo em dispositivos Android e emular diferentes configurações de hardware e sistema operacional.
- **Xcode**: Empregado no desenvolvimento e testes para dispositivos iOS, aproveitando o simulador de iPhone.

Bibliotecas Principais

- **React Native** (v0.74.5): Framework base para o desenvolvimento do aplicativo.
- **React** (v 18.2.0): Biblioteca JavaScript para construção de interfaces de usuário.

Gerenciamento de Navegação

- **@react-navigation/native** (v6.1.18): Para implementar a navegação entre as telas do aplicativo.
- **@react-navigation/native-stack** (v6.11.0): Stack navigator utilizado para a transição entre telas de forma fluida.

Comunicação com a API

- **Axios** (v1.7.7): Biblioteca para consumo de APIs RESTful, facilitando as requisições HTTP (GET, POST, PUT, DELETE).
- **json-server** (v1.0.0-beta.3): Utilizado para simular um backend e gerenciar os dados do aplicativo durante o desenvolvimento.

Interface do Usuário

- **@react-native-picker/picker** (v2.9.0): Fornece um componente de seleção (drop down) nativo para escolher opções, como categorias ou atributos dos produtos.
- **expo-status-bar** (v1.12.1): Utilizado para personalizar a barra de status do sistema.
- **react-native-safe-area-context** (v4.14.0): Garante que os componentes respeitem as áreas seguras do dispositivo.
- **react-native-screens** (v3.35.0): Otimiza a performance das transições de telas no aplicativo.

Ferramentas de Desenvolvimento

- **@babel/core** (v7.20.0): Configurado para transpirar o código, garantindo compatibilidade com diversas versões de JavaScript.

Resumo

O **iStore** foi construído utilizando ferramentas modernas e práticas, que permitem ao aplicativo oferecer uma experiência fluida e eficiente para os usuários. O uso do Expo garantiu rapidez na configuração e testes, enquanto as bibliotecas instaladas via NPM adicionam funcionalidades essenciais para navegação, interface e comunicação com a API.

3. Estrutura do Projeto

O **iStore** foi desenvolvido utilizando uma estrutura modular, visando organização e reutilização de código. Abaixo, detalhamos os principais diretórios e arquivos do projeto:

Arquivos Principais

- **db.json**: Banco de dados simulado utilizando o `json-server` para o gerenciamento de produtos.
- **Main Navigation.js**: Arquivo principal responsável por configurar as rotas de navegação do aplicativo.

Diretório: `src`

Contém os recursos principais do projeto, organizados em subdiretórios conforme sua funcionalidade:

1. **`src/assets`**:
   Este diretório armazena os recursos visuais utilizados no aplicativo.
   - **Icons**: Ícones usados na interface do usuário, como:
     - `Apple.png`, `arrow_left.png`, `edit.png`, `trash.png`, entre outros.
   - **Images**: Imagens gerais do aplicativo, incluindo:
     - `Banner.png`: Usado na página inicial.
     - `logo-splash.png`: Exibido na tela de splash.
2. **`src/assets/styles`**:
   Estilos personalizados para páginas específicas, organizados por arquivo, como:
   - `CriarProdutoStyle.js`: Estilos para a página de criação de produtos.
   - `EditarProdutoStyle.js`: Estilos para a edição de produtos.

Diretório: `src/Components`

Armazena componentes reutilizáveis que tornam o código mais modular:

- **Banner.js**: Componente que exibe o banner principal.
- **Feed.js**: Gerencia o feed de produtos.

- **Header.js**: Cabeçalho padrão usado em diferentes telas.

Diretório: `src/Pages`

Contém as páginas principais do aplicativo:

- **CriarProduto.js**: Página para adicionar um novo produto ao estoque.
- **DeletarProduto.js**: Tela para excluir produtos.
- **DetalhesProduto.js**: Exibe informações detalhadas de um produto.
- **EditarProduto.js**: Permite editar as informações de um produto existente.
- **Home.js**: Página inicial do aplicativo.
- **Profile.js**: Tela de perfil do usuário.
- **Splash.js**: Tela inicial exibida ao abrir o aplicativo.
- **VizualizarEstoque.js**: Tela para listar todos os produtos disponíveis no estoque.

4. Funcionalidades do Aplicativo

Exibir Lista de Produtos

Essa funcionalidade, implementada na tela `VizualizarEstoque.js`, permite aos usuários visualizar os produtos disponíveis no estoque, organizados por categorias.

Fluxo da Funcionalidade:

1. **Seleção de Categorias:**
   - O aplicativo suporta quatro categorias principais: **Mac**, **iPhone**, **iPad** e **Watch**.
   - Cada categoria é representada por ícones. Quando uma categoria é selecionada, os produtos correspondentes são carregados.
2. **Carregamento de Produtos:**
   - Uma requisição **GET** é feita ao servidor local (simulado via `json-server`) usando a URL correspondente à categoria ativa.
   - Os produtos retornados são armazenados no estado `produtos`.
3. **Renderização de Produtos:**
   - Os itens são exibidos em uma lista interativa utilizando o componente `FlatList`, mostrando informações como:
     - Nome
     - Modelo

        - Preço
        - Cores disponíveis (caso aplicável).
    - Produtos sem imagens específicas exibem uma imagem padrão.
4. **Interação com Produtos:**
    - Ao clicar em um produto, o usuário é redirecionado para a página de **Detalhes do Produto**, onde mais informações estão disponíveis.

Componentes Principais Utilizados:

- `FlatList`: Para renderizar eficientemente listas de produtos.
- `TouchableOpacity`: Para permitir interação com os itens da lista.
- `Alert`: Para exibir mensagens caso não sejam encontrados produtos na categoria selecionada.

Estados do Componente:

- `categoriaAtiva`: Indica a categoria de produtos atualmente selecionada. Por padrão, inicia com **Mac**.
- `produtos`: Armazena a lista de produtos carregados da categoria ativa.
- `loading`: Gerencia o estado de carregamento enquanto os dados estão sendo obtidos.

Funções Relevantes:

- `listarProdutos`:
    - Objetivo: Fazer uma requisição ao servidor para carregar os produtos da categoria ativa.
    - Endpoint: `http://10.0.2.2:3000/{categoriaAtiva}`.
    - Fluxo:
        - Armazena os produtos retornados no estado `produtos` ou exibe um alerta em caso de falha.

Exemplo de Navegação:

1. O usuário seleciona a categoria **iPhone**.
2. A função `listarProdutos` é acionada para carregar os itens correspondentes.
3. Os produtos são exibidos em cartões interativos.
4. Ao clicar em um produto, o usuário é redirecionado para a página **Detalhes do Produto**.

Adicionar Produto

Essa funcionalidade, implementada na tela `CriarProduto.js`, permite ao usuário cadastrar novos produtos no sistema. Os produtos são organizados em categorias específicas e incluem atributos como nome, modelo, ano, preço, cores e opções de armazenamento.

Fluxo da Funcionalidade:

1. **Seleção de Categorias:**
    - O usuário escolhe a categoria para o novo produto entre: **Mac**, **iPhone**, **iPad** e **Watch**.
    - Cada categoria é representada por um ícone. Ao clicar em um ícone, a categoria ativa é atualizada.
2. **Preenchimento do Formulário:**
    - Campos obrigatórios:
        - Nome
        - Modelo
        - Ano
        - Preço
        - URL da Imagem
    - Campos adicionais:
        - Lista de Cores: O usuário pode adicionar cores preenchendo nome e valor.
        - Lista de Armazenamento: Permite adicionar diferentes capacidades e preços.
3. **Validação e Cadastro:**
    - Após preencher o formulário, o usuário clica em **Cadastrar Produto**.
    - A função `handleSubmitProduto` valida os dados e envia uma requisição **POST** ao servidor.
    - Em caso de sucesso:
        - O produto é adicionado à lista da categoria ativa.
        - O formulário é resetado, e o usuário recebe uma mensagem de confirmação.

Componentes Principais Utilizados:

- **`ScrollView`**: Para permitir a rolagem em formulários extensos.
- **`TextInput`**: Para entrada de texto nos campos do formulário.
- **`TouchableOpacity`**: Para botões interativos, como adicionar cores ou armazenamento.

- **`ActivityIndicator`**: Para exibir uma animação de carregamento enquanto o produto é cadastrado.

Estados do Componente:

- `categoriaAtiva`: Define a categoria selecionada (ex.: `/mac` ou `/iphone`).
- `loading`: Indica se uma operação está em andamento (como salvar o produto).
- `produtos`: Armazena a lista de produtos da categoria ativa.
- `novoProduto`: Objeto que contém os dados do novo produto a ser cadastrado, incluindo:
    - Nome, Modelo, Ano, Preço, URL da Imagem
    - ListaCores (array de cores)
    - ListaArmazenamentos (array de opções de armazenamento)

Funções Relevantes:

- **`listarCategorias(categoriaUrl)`**:
    - Objetivo: Atualiza a lista de produtos da categoria selecionada.
    - Endpoint: `http://10.0.2.2:3000{categoriaUrl}`.
- **`handleSubmitProduto()`**:
    - Objetivo: Envia os dados do produto ao servidor e atualiza a lista da categoria ativa.
- **`addCor()`**:
    - Adiciona um novo campo de cor ao formulário.
- **`addArmazenamento()`**:
    - Adiciona um novo campo de armazenamento ao formulário.
- **`handleInputChange(name, value)`**:
    - Atualiza os valores dos campos no estado `novoProduto`.

Exemplo de Navegação e Uso:

1. O usuário seleciona a categoria **Watch**.
2. Preenche os campos obrigatórios e adiciona duas cores e três opções de armazenamento.
3. Clica em **Cadastrar Produto**.
4. O produto é salvo no servidor e exibido na lista de produtos da categoria.

Editar Produto

Essa funcionalidade, implementada na tela `EditarProduto.js`, permite ao usuário buscar um produto existente pelo ID, atualizar suas informações e salvar as alterações no sistema.

Fluxo da Funcionalidade:

1. **Seleção de Categorias:**
    - O usuário escolhe a categoria do produto entre: **Mac**, **iPhone**, **iPad** e **Watch**.
    - Cada categoria é representada por um ícone. Ao selecionar uma categoria, a URL ativa é ajustada para buscar e editar produtos dessa categoria.
2. **Busca do Produto:**
    - O usuário insere o **ID do produto** e clica em **Buscar Produto**.
    - A função `buscarProduto` realiza uma requisição **GET** para obter os dados do produto a partir do servidor.
    - Os dados são carregados no formulário para edição.
3. **Edição e Atualização:**
    - Campos disponíveis para edição:
        - Nome
        - Modelo
        - Ano
        - Preço
        - URL da Imagem
        - Cores (nome e valor)
        - Opções de Armazenamento (capacidade e preço)
    - O usuário pode adicionar novas cores ou opções de armazenamento dinamicamente.
    - Após realizar as alterações, o usuário clica em **Salvar Alterações**.
    - A função `handleSubmitProduto` realiza uma requisição **PUT** para atualizar o produto no servidor.

Componentes Principais Utilizados:

- **`ScrollView`**: Permite rolagem em formulários extensos.
- **`TextInput`**: Para entrada de texto nos campos do formulário.
- **`TouchableOpacity`**: Para botões interativos, como buscar, adicionar ou salvar.
- **`ActivityIndicator`**: Exibe uma animação de carregamento durante a busca ou edição do produto.

Estados do Componente:

- `categoriaAtiva`: Define a categoria do produto a ser editado.
- `idProduto`: Armazena o ID do produto inserido pelo usuário.
- `produtoEditado`: Objeto contendo os dados do produto para edição:
  - Nome, Modelo, Ano, Preço, URL da Imagem
  - ListaCores (array de cores)
  - ListaArmazenamentos (array de opções de armazenamento)
- `loading`: Indica se a operação de busca ou edição está em andamento.

Funções Relevantes:

- **`buscarProduto(id)`**:
  - Objetivo: Busca os dados do produto pelo ID fornecido.
  - Endpoint: `http://10.0.2.2:3000{categoriaAtiva}/{id}`.
  - Fluxo:
    - Caso encontre o produto, carrega as informações no estado `produtoEditado`.
    - Exibe uma mensagem de erro caso o ID seja inválido ou a conexão falhe.
- **`handleSubmitProduto()`**:
  - Objetivo: Envia as alterações ao servidor para atualizar o produto.
  - Endpoint: `http://10.0.2.2:3000{categoriaAtiva}/{idProduto}`.
- **`addCor()`**:
  - Adiciona um novo campo de cor ao formulário.
- **`addArmazenamento()`**:
  - Adiciona um novo campo de armazenamento ao formulário.
- **`handleInputChange(name, value)`**:
  - Atualiza os valores dos campos no estado `produtoEditado`.

Exemplo de Navegação e Uso:

1. O usuário seleciona a categoria **iPhone** e insere o **ID do produto**.
2. Clica em **Buscar Produto** para carregar as informações.
3. Altera o modelo, adiciona uma nova cor e uma nova opção de armazenamento.
4. Clica em **Salvar Alterações** para atualizar o produto no servidor.
5. Após a edição, o usuário é redirecionado para a página anterior com as alterações salvas.

Visualizar Produtos

A funcionalidade de **Visualizar Produtos**, implementada na tela `VisualizarProdutos.js`, permite aos usuários navegar por diferentes categorias e visualizar os produtos disponíveis no estoque, exibindo informações detalhadas de cada item.

Fluxo da Funcionalidade:

1. **Seleção de Categorias:**
    - As categorias disponíveis são: **Mac**, **iPhone**, **iPad** e **Watch**.
    - Cada categoria é representada por um ícone. Quando uma categoria é selecionada, a lista de produtos correspondente é carregada automaticamente.
2. **Carregamento dos Produtos:**
    - A função `listarProdutos` realiza uma requisição **GET** ao servidor para buscar os produtos da categoria ativa.
    - A resposta do servidor é validada para garantir que apenas produtos válidos sejam exibidos.
    - Caso nenhum produto seja encontrado:
        - Um alerta é exibido para o usuário.
        - A lista é definida como vazia.
3. **Exibição dos Produtos:**
    - Os produtos são renderizados em uma lista interativa utilizando o componente `FlatList`, com informações como:
        - Nome
        - Modelo
        - Preço
        - ID
        - Cores disponíveis (exibidas como círculos coloridos).
    - Produtos sem imagem exibem uma mensagem padrão indicando a ausência de imagem.
4. **Interação com Produtos:**
    - Ao clicar em um produto, o usuário é redirecionado para a tela de **Detalhes do Produto**, onde pode visualizar informações mais detalhadas.

Componentes Principais Utilizados:

- **`FlatList`**: Para renderizar a lista de produtos de forma eficiente.
- **`TouchableOpacity`**: Para tornar os itens da lista interativos.
- **`Alert`**: Para exibir mensagens caso não sejam encontrados produtos na categoria selecionada.
- **`Image`**: Para exibir as imagens dos produtos ou uma imagem padrão quando ausente.

Estados do Componente:

- `categoriaAtiva`: Indica a categoria de produtos atualmente selecionada.
- `produtos`: Armazena a lista de produtos carregados da categoria ativa.
- `loading`: Controla o estado de carregamento enquanto os dados estão sendo obtidos.

Funções Relevantes:

- **`listarProdutos(categoriaUrl)`**:
  - Objetivo: Realizar uma requisição ao servidor para buscar produtos da categoria selecionada.
  - Endpoint: `http://10.0.2.2:3000{categoriaUrl}`.
  - Fluxo:
    - Caso existam produtos válidos, eles são exibidos.
    - Em caso de erro ou ausência de produtos, a lista é definida como vazia, e uma mensagem é exibida ao usuário.
- **`CardProdutos({ item })`**:
  - Renderiza cada produto individualmente, exibindo suas informações principais.
  - Inclui lógica para lidar com produtos que não possuem imagens ou cores disponíveis.

Exemplo de Navegação e Uso:

1. O usuário seleciona a categoria **Mac**.
2. A lista de produtos da categoria é carregada e exibida em forma de cartões.
3. O usuário clica em um produto, sendo redirecionado para a tela **Detalhes do Produto** para mais informações.

5. Fluxo de Navegação

Navegação no Aplicativo:

1. **Tela Inicial:**
    - Quando o aplicativo é aberto, o usuário visualiza a **tela de splash** (`SplashScreen`) com o logotipo.
    - Após a exibição da splash screen, o usuário é redirecionado automaticamente para a **página inicial** (`Home`).
2. **Estrutura de Navegação:**
    - A navegação é gerenciada pelo `react-navigation`, utilizando o `Native Stack Navigator`.
    - Cada funcionalidade do aplicativo está mapeada como uma **rota**, facilitando o acesso a diferentes telas.
3. **Caminho entre Funcionalidades:**
    - **Home:** Página principal que exibe as opções de gerenciamento.
        - Botões para criar, visualizar, editar ou excluir produtos.
    - **Perfil:** Tela de perfil do usuário (`Profile`).
    - **Visualizar Estoque:** Lista de produtos organizados por categorias (`VizualizarEstoque`).
    - **Criar Produto:** Formulário para adicionar novos produtos ao estoque (`CriarProduto`).
    - **Editar Produto:** Tela para editar informações de um produto existente (`EditarProduto`).
    - **Excluir Produto:** Tela para exclusão de produtos selecionados (`DeletarProduto`).
    - **Detalhes do Produto:** Mostra informações detalhadas de um item do estoque (`DetalhesProduto`).

Descrição das Rotas:

- **`Splash`:**
    - Primeira tela exibida ao abrir o app.
    - Redireciona automaticamente para a **Home**.
- **`Home`:**
    - Página inicial com acesso às principais funcionalidades.
- **`Profile`:**
    - Tela de perfil do usuário (oculta o cabeçalho personalizado).
- **`CriarProduto`:**
    - Tela para cadastro de novos produtos.

- **`EditarProduto`:**
  - Tela para edição de produtos existentes.
- **`FeedEstoque`:**
  - Tela para visualizar e navegar pelo estoque.
- **`DeletarProduto`:**
  - Tela para exclusão de produtos.
- **`DetalhesProduto`:**
  - Tela para exibir as informações completas de um produto (oculta o cabeçalho).

Navegação entre Telas:

- A navegação é realizada utilizando a API `navigation.navigate("NomeDaRota")`, que facilita a transição entre as telas.

Exemplo de navegação para editar um produto:

```
<TouchableOpacity onPress={() => navigation.navigate("EditarProduto", { id: produto.id })}>
  <Text>Editar Produto</Text>
</TouchableOpacity>
```

6. API

Configuração do Servidor:

O aplicativo utiliza o **json-server** para simular uma API RESTful. A configuração é feita por meio do arquivo `db.json`, onde estão definidos os dados relacionados às categorias de produtos (Mac, iPhone, iPad, Watch). O json-server gera automaticamente os endpoints para cada objeto presente no arquivo JSON.

Passos para Configuração:

1. **Instalar o json-server**:

Certifique-se de que o json-server está instalado globalmente ou no projeto:
`npm install -g json-server`

2. **Inicializar o Servidor**:

Use o seguinte comando para iniciar o json-server apontando para o arquivo `db.json`:
`json-server --watch db.json --port 3000`

3. **Estrutura de Dados (db.json)**:
    - O arquivo contém objetos organizados por categorias:
        - **mac**
        - **iphone**
        - **ipad**
        - **watch**
    - Cada produto tem atributos como `nome`, `modelo`, `ano`, `preco`, `ListaCores` (array de cores) e `ListaArmazenamentos` (array de opções de armazenamento).

Endpoints Disponíveis:

- **`GET /categoria`**:
    - Retorna todos os itens de uma categoria.
    - Exemplo: `/mac`, `/iphone`.
- **`POST /categoria`**:
    - Adiciona um novo item à categoria.
    - Exemplo: `/mac`.
- **`GET /categoria/:id`**:
    - Retorna um item específico pelo ID.
    - Exemplo: `/mac/6e1e`.
- **`PUT /categoria/:id`**:
    - Atualiza um item existente pelo ID.
    - Exemplo: `/iphone/8cc8`.
- **`DELETE /categoria/:id`**:
    - Remove um item da categoria pelo ID.
    - Exemplo: `/ipad/fa64`.

Exemplo de Payloads:

Adicionar Produto (`POST /mac`):

```
{
  "nome": "MacBook Pro",
  "modelo": "M3 2024",
  "ano": 2024,
  "preco": 1999.99,
  "ListaCores": [
    {
      "nomeCor": "Preto",
      "valorCor": "#000000"
    }
  ],
  "ListaArmazenamentos": [
    {
      "capacidade": "256GB",
      "preco": 1999.99
    }
  ]
}
```

1.

Atualizar Produto (`PUT /mac/6e1e`):

```
{
  "nome": "MacBook Air",
  "modelo": "M2 2024",
  "ano": 2024,
  "preco": 1099.99,
  "ListaCores": [
    {
      "nomeCor": "Cinza Espacial",
      "valorCor": "#1D1F20"
    }
  ],
  "ListaArmazenamentos": [
    {
      "capacidade": "512GB",
      "preco": 1249.99
    }
  ]
}
```

2. **Deletar Produto (`DELETE /iphone/8cc8`)**:
    - Este endpoint não requer corpo na requisição.

Observações:

- O servidor atualiza os dados automaticamente no arquivo `db.json` quando ações de escrita (POST, PUT, DELETE) são realizadas.
- O campo `id` é gerado automaticamente pelo json-server caso não seja fornecido.

Vantagens do json-server:

- Simula uma API completa rapidamente, ideal para testes em desenvolvimento.
- Possui suporte para filtro de dados em endpoints utilizando query parameters. Exemplo:
    - `GET /mac?ano=2023`: Retorna todos os Macs lançados em 2023.

7. Interface do Usuário

O design do aplicativo foi inspirado no modelo **Apple Store Mobile App UI Kit**, disponível no Figma, com foco em uma experiência visual premium e alinhada com o padrão estético da marca Apple.

Paleta de Cores

- **Primárias**: Preto e branco para fundo e textos.
- **Acentos**: Tons sutis de cinza e cores características dos produtos (exemplo: dourado, prateado e cinza espacial).

Layout

- **Tela Inicial**: Inclui um banner promocional no topo, categorias (Mac, iPhone, iPad, Watch) em ícones e uma lista de produtos exibida em formato de cartões.
- **Detalhes do Produto**: Cada produto conta com uma página específica que destaca a imagem do item, opções de cor, armazenamento e preço, além de um botão de compra.

- **Inspiração**

O design foi diretamente adaptado e ajustado com base no **Free Apple Store Mobile App UI Kit** do Figma, oferecendo uma navegação intuitiva e moderna que reflete o conceito visual da loja oficial da Apple.

Link para o modelo de inspiração: Apple Store UI Kit no Figma.

8. Conclusão

O desenvolvimento do aplicativo iStore representou uma experiência enriquecedora e desafiadora, que exigiu a aplicação de conceitos fundamentais de desenvolvimento mobile, como navegação entre telas, gerenciamento de estado, consumo de APIs e design responsivo para dispositivos móveis. Durante o processo, fomos capazes de explorar e implementar soluções práticas que facilitam o gerenciamento de estoque em lojas especializadas na venda de produtos Apple.

A construção do iStore utilizou tecnologias modernas como React Native e Expo, permitindo um desenvolvimento mais ágil e a simulação de um backend com o uso do json-server. Com isso, conseguimos criar uma aplicação que oferece uma experiência fluida e eficiente para os usuários, com funcionalidade de controle de inventário acessível diretamente do celular.

A estrutura modular do projeto, com componentes reutilizáveis e uma lógica bem definida, permitiu manter a organização do código e a facilidade de manutenção ao longo do desenvolvimento. As funcionalidades implementadas, como a visualização de produtos, adição, edição e exclusão de produtos no estoque, proporcionaram uma gestão otimizada e melhoraram a produtividade da equipe.

Além disso, o feedback dos usuários e a experiência prática adquirida durante a implementação do iStore mostraram que soluções mobile para o gerenciamento de estoque são essenciais para o mercado atual, principalmente em empresas que comercializam produtos de alto valor agregado, como os dispositivos Apple. A integração de funcionalidades como navegação intuitiva, design responsivo e consumo de APIs foi crucial para atender às necessidades dos usuários e proporcionar uma experiência de usuário final de qualidade.

Em resumo, o desenvolvimento do aplicativo iStore não só demonstrou a eficácia das tecnologias móveis no gerenciamento de estoque, mas também ressaltou a importância da acessibilidade e da eficiência operacional para pequenas e médias empresas que buscam modernizar seus processos de gestão de inventário.