

OPRYT

Um breve manual de confecção



CTII417 - 2023

Gabriel Henrique da Cruz Pereira Gonçalves

Gilmar dos Santos Diniz

João Felipe da Silva

Lucas da Silva Santos

Murilo Paulino da Silva

Sumário

1. INTRODUÇÃO	2
2. ABORDAGENS TÉCNICAS GERAIS	3
3. JAVASCRIPT E TYPESCRIPT	4
3.1 Visual Studio Code	4
4. REACT NATIVE	7
5. NODE.JS	9
6. MONGO DB	11
7. TENSORFLOW.JS	13
8. VISUALIZAÇÃO DO APLICATIVO	15
8.1 Visualização da página <i>web</i>	20
REFERÊNCIAS BIBLIOGRÁFICAS	23

I. INTRODUÇÃO

O projeto proposto foi a confecção de um aplicativo para *smartphones* que funcionasse como uma espécie de gerenciador e organizador das fotografias presentes na galeria do dispositivo. Na aplicação, a qual está unida a uma inteligência artificial, o usuário seleciona imagens aleatórias da galeria do seu dispositivo eletrônico e o sistema elaborado se responsabiliza de criar pastas com categorias que se relacionam com o conteúdo das imagens selecionadas pelo usuário e arquivá-las nestas pastas. Além do processo automatizado, o usuário também pode realizar o procedimento de forma manual.

Para que o funcionamento efetivo do aplicativo elaborado ocorra, necessita-se de uma conexão constante com a internet para que a inteligência artificial conclua seu trabalho e que as imagens e os dados informados fiquem gravados no banco de dados do servidor do sistema. Através disso, gera-se a possibilidade de que uma conta seja acessada em distintos dispositivos, isso se um *login* for realizado.

Com isso, a aplicação proposta apresenta uma solução para a organização das imagens presentes na galeria do celular do usuário, já que muitas vezes as imagens são armazenadas sem nenhum tipo de classificação e misturadas umas com as outras na galeria dos *smartphones* de quase todos.

2. ABORDAGENS TÉCNICAS GERAIS

Para a realização do projeto, foi feito um levantamento em diversas bibliografias digitais com o objetivo de encontrar as linguagens de programação e as ferramentas mais adequadas para a criação do aplicativo.

Como resultado, optou-se utilizar as linguagens *JavaScript* e *TypeScript* devido as suas amplas adoções no mercado e suportes disponíveis na comunidade de desenvolvimento. O *JavaScript* é uma linguagem de programação versátil, que junto a um *framework*/biblioteca é possível elaborar aplicações nativas de determinados sistemas (NOÇÕES, [s.d.]). Já o *TypeScript*, uma linguagem de extensão do *JavaScript*, oferece tipagem estática e recursos adicionais que facilitam a detecção de erros e o desenvolvimento em larga escala, tornando o código mais seguro e legível (TROQUATTE, [s.d.]).

O *React Native* foi escolhido como *framework* para o desenvolvimento do aplicativo. Com o *React Native*, é possível criar aplicativos nativos para *iOS* e *Android* usando uma única base de código, o que economiza tempo e recursos. Além disso, a escolha do *Expo*, que é uma plataforma e conjunto de ferramentas para *React Native*, simplifica o processo de desenvolvimento, fornecendo recursos como bibliotecas de componentes prontos para uso e, assim, facilitando a compilação e distribuição do aplicativo (CUNHA, 2023).

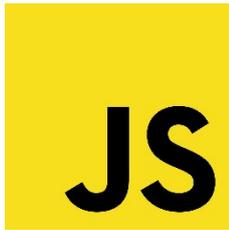
Para o lado do servidor, foi optado o *Node.js* como plataforma *back-end* e o *MongoDB* como banco de dados. O *Node.js* é conhecido por sua escalabilidade e eficiência, permitindo que o aplicativo lide com uma grande quantidade de solicitações simultâneas (SANTOS, 2023). Ele também é baseado em *JavaScript*, o que simplifica a troca de código entre o *front-end* e o *back-end*. O *MongoDB*, é um banco de dados *NoSQL* flexível e altamente escalável, que se integra bem com o *Node.js* (FRANCISCATO, 2023).

Para a tarefa de reconhecimento e classificação de imagens, escolheu-se o *Tensorflow.js*, uma biblioteca de aprendizado de máquina em *JavaScript*. O *Tensorflow.js* permite que se realize o treinamento e a inferência de modelos de aprendizado de máquina diretamente no navegador ou no ambiente *Node.js* (TENSORFLOW, [s.d.]). Isso significa que se pode implementar algoritmos de classificação de imagens utilizando redes neurais convolucionais (CNNs, o que significa *Convolutional Neural Networks* na língua inglesa), um tipo de arquitetura de rede neural especializada em processamento de imagens.

3. JAVASCRIPT E TYPESCRIPT

As linguagens de programação *JavaScript* e *TypeScript* foram utilizadas como base de todo o projeto confeccionado, servindo como pontes para serem feitas as conexões com as demais estruturas [banco de dados, interfaces gráficas e servidor].

Figura 1. Logotipo da linguagem de programação *JavaScript*.



Fonte:

<https://pt.wikipedia.org/wiki/JavaScript>

Figura 2. Logotipo da linguagem de programação *TypeScript*.



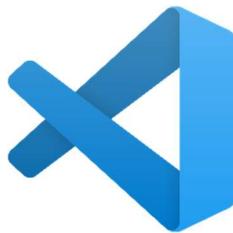
Fonte:

[https://pt.m.wikipedia.org/wiki/Ficheiro:Typescript logo 2020.svg](https://pt.m.wikipedia.org/wiki/Ficheiro:Typescript_logo_2020.svg)

3.1 Visual Studio Code

A IDE (Integrated Development Environment, que significa Ambiente de Desenvolvimento Integrado em português) utilizada para a escrita do código dessas linguagens foi o *Visual Studio Code*, da *Microsoft*.

Figura 3. Logotipo da IDE *Visual Studio Code*.

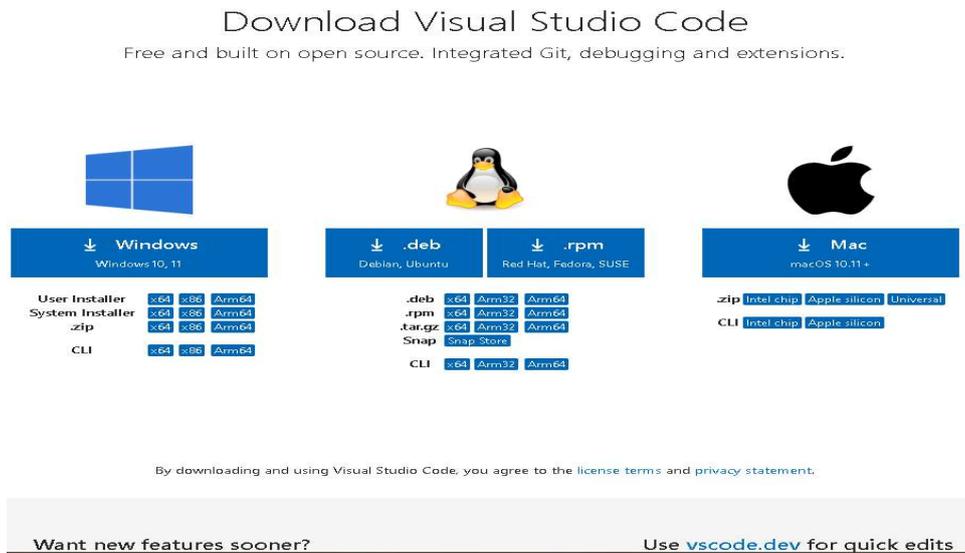


Fonte:

https://az.wikipedia.org/wiki/Visual_Studio_Code

O *Visual Studio Code* se trata de uma IDE gratuita, que pode ser instalada em qualquer *desktop* através da *web*, como ilustra a imagem a seguir.

Figura 4. Print de tela da fase inicial de instalação do *Visual Studio Code*.

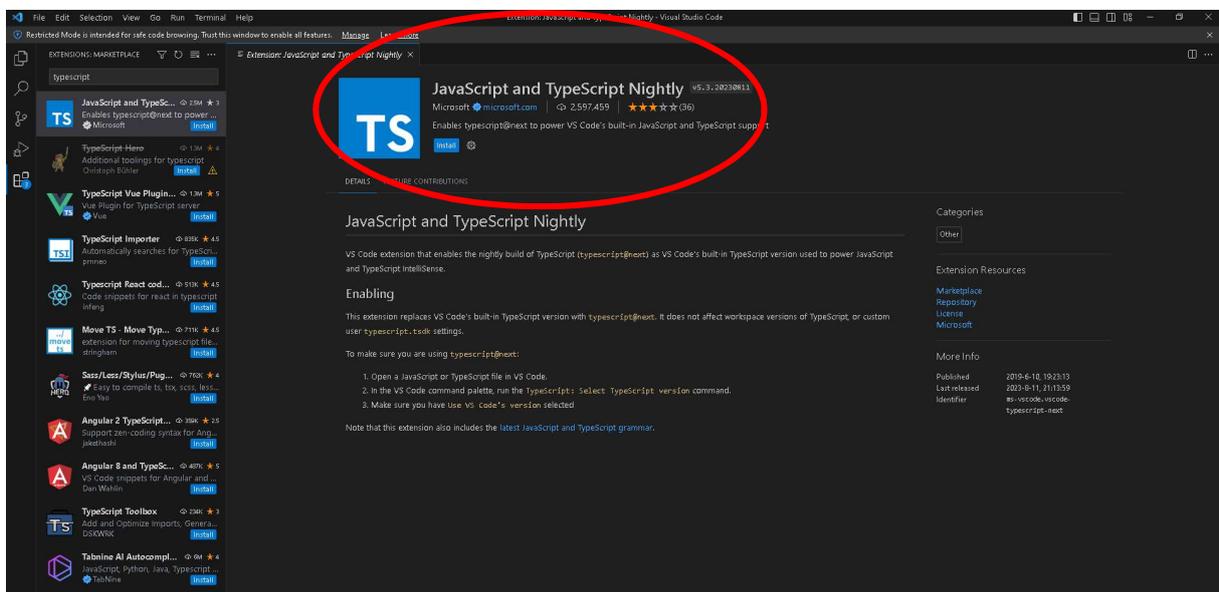


Fonte: print de autoria própria, página disponível em

<<https://code.visualstudio.com/download>>.

Após o *download* ser concluído, necessita-se da instalação da extensão *JavaScript and TypeScript Nightly* para que as linguagens sejam inseridas no *Visual Studio Code*.

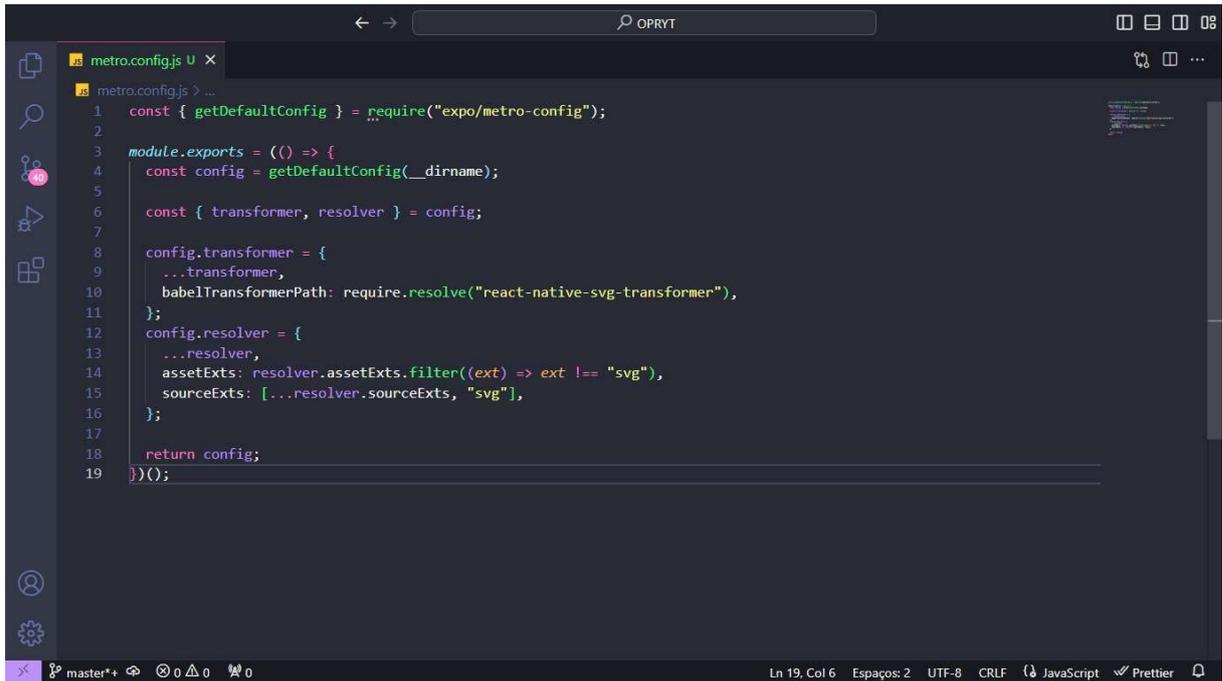
Figura 5. Print indicando a extensão que deve ser instalada.



Fonte: print de autoria própria.

Feito isso, todo processo de programação foi iniciado; alguns trechos de código são exibidos nas imagens a seguir.

Figura 6. *Print* de parte do código escrito na linguagem *JavaScript*.



```
metro.config.js U X
metro.config.js > ...
1  const { getDefaultConfig } = require("expo/metro-config");
2
3  module.exports = (() => {
4    const config = getDefaultConfig(__dirname);
5
6    const { transformer, resolver } = config;
7
8    config.transformer = {
9      ...transformer,
10     babelTransformerPath: require.resolve("react-native-svg-transformer"),
11   };
12   config.resolver = {
13     ...resolver,
14     assetExts: resolver.assetExts.filter((ext) => ext !== "svg"),
15     sourceExts: [...resolver.sourceExts, "svg"],
16   };
17
18   return config;
19 })();
```

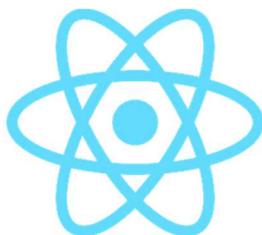
Ln 19, Col 6 Espaços: 2 UTF-8 CRLF JavaScript Prettier

Fonte: *print* de autoria própria.

4. REACT NATIVE

Para desenvolver a parte gráfica do aplicativo, utilizou-se as ferramentas do *React Native*. Para que isso fosse possível, foi necessário instalar sua extensão no *Visual Studio Code*, a qual é exibida no próximo *print*.

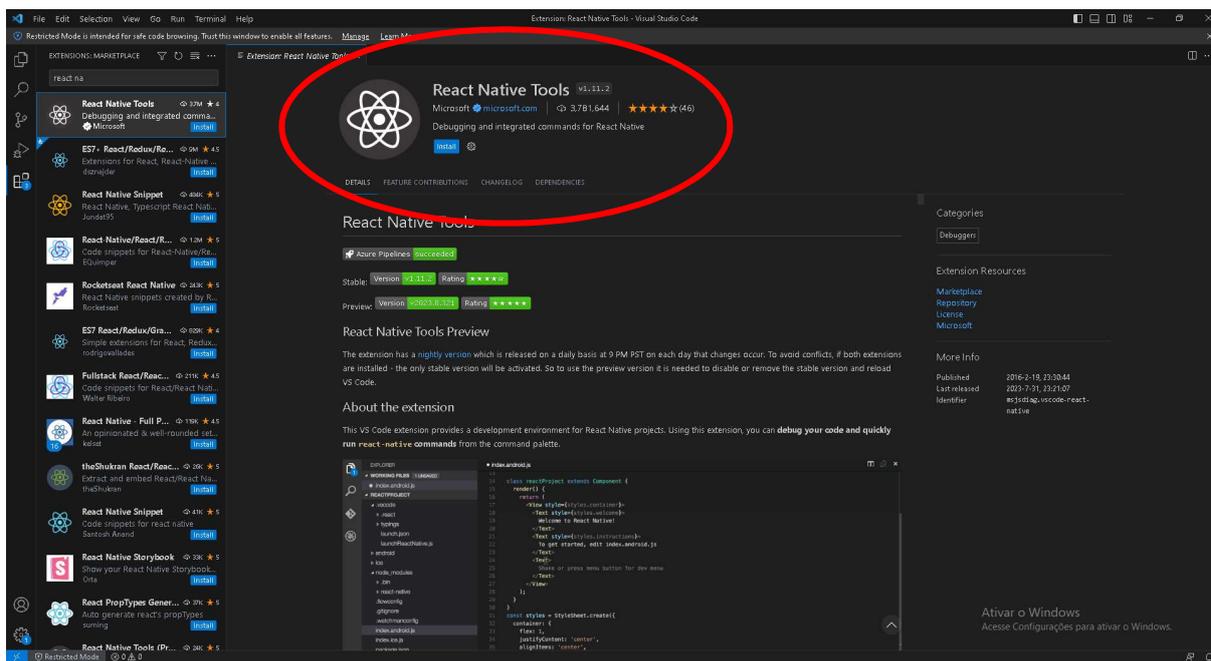
Figura 7. Logotipo do *React Native*.



Fonte:

https://en.wikipedia.org/wiki/React_Native

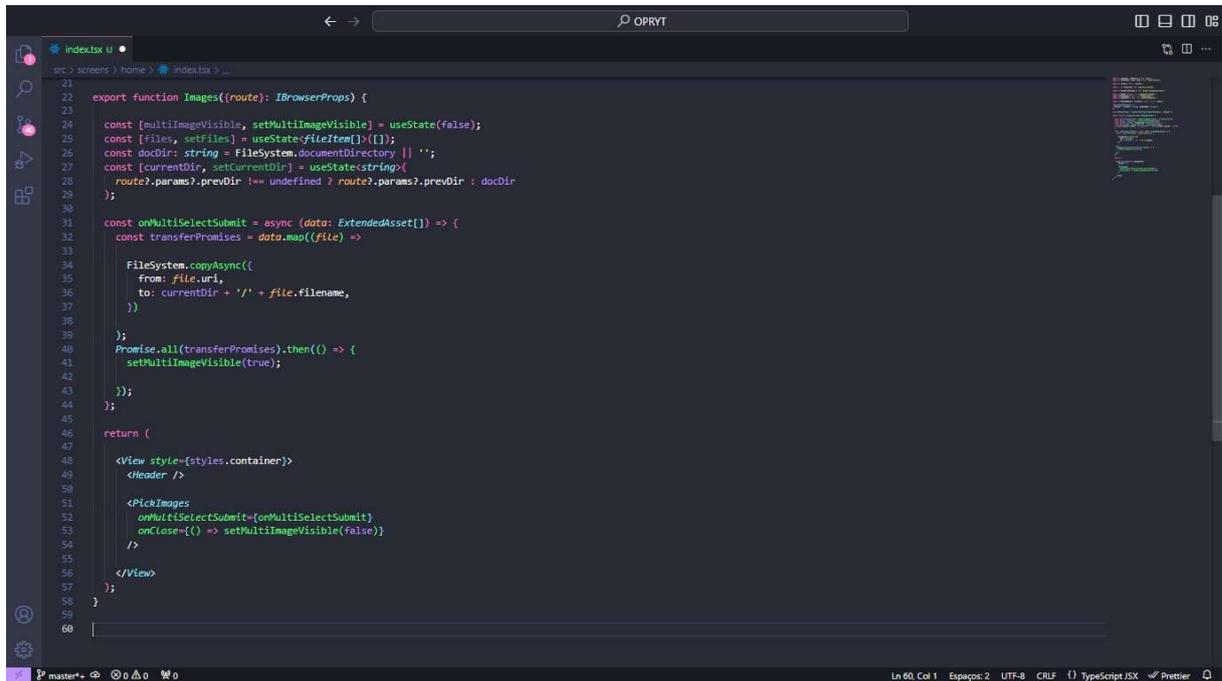
Figura 8. *Print* indicando a extensão que deve ser instalada.



Fonte: *print* de autoria própria.

Feito isso, o processo de confecção foi iniciado, alguns trechos de código são exibidos nas imagens a seguir.

Figura 9. *Print* indicando a construção do álbum de fotografias do usuário utilizando o *React Native*.



```
21
22 export function Images({route}: IBrowserProps) {
23
24   const [multiImageVisible, setMultiImageVisible] = useState(false);
25   const [files, setFiles] = useState<fileItem[]>([]);
26   const docDir: string = FileSystem.documentDirectory || '';
27   const [currentDir, setCurrentDir] = useState<string>({
28     route?.params?.prevDir !== undefined ? route?.params?.prevDir : docDir
29   });
30
31   const onMultiSelectSubmit = async (data: ExtendedAsset[]) => {
32     const transferPromises = data.map((file) => {
33
34       FileSystem.copyAsync({
35         from: file.uri,
36         to: currentDir + '/' + file.filename,
37       })
38     });
39
40     Promise.all(transferPromises).then(() => {
41       setMultiImageVisible(true);
42     });
43   });
44 };
45
46 return (
47
48   <View style={styles.container}>
49     <Header />
50
51     <PickImages
52       onMultiSelectSubmit={onMultiSelectSubmit}
53       onClose={() => setMultiImageVisible(false)}
54     />
55
56   </View>
57 );
58 }
59
60
```

Fonte: *print* de autoria própria.

5. NODE.JS

Para o *back-end* do servidor, foi optada a plataforma *Node.js*. Visando sua utilização, foi necessária instalação de sua extensão no *Visual Studio Code*, a qual é exibida no próximo *print*.

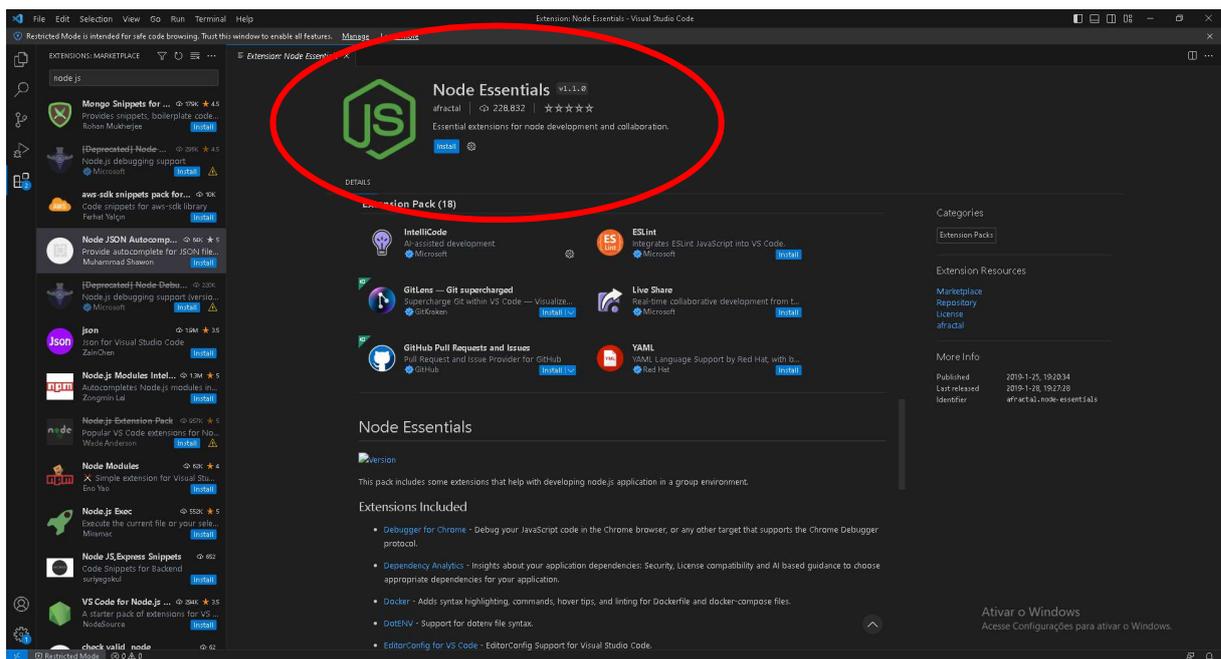
Figura 10. Logotipo do *Node.js*.



Fonte:

<https://pt.wikipedia.org/wiki/Node.js>

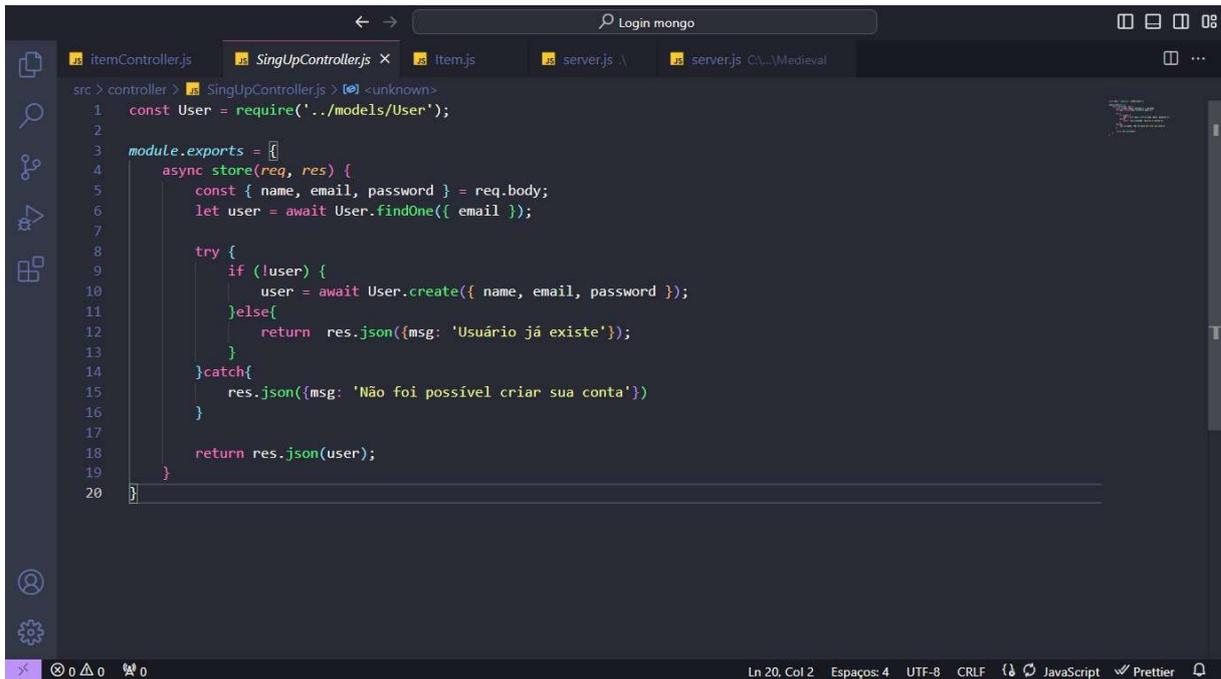
Figura 11. *Print* indicando a extensão que deve ser instalada.



Fonte: *print* de autoria própria.

Posto isso, o processo de programação do servidor foi iniciado.

Figura 12. Trecho de código em *Node.js*.



```
src > controller > SingUpController.js > <unknown>
1  const User = require('../models/User');
2
3  module.exports = {
4    async store(req, res) {
5      const { name, email, password } = req.body;
6      let user = await User.findOne({ email });
7
8      try {
9        if (!user) {
10         user = await User.create({ name, email, password });
11        } else {
12         return res.json({msg: 'Usuário já existe'});
13        }
14      } catch {
15        res.json({msg: 'Não foi possível criar sua conta'})
16      }
17
18      return res.json(user);
19    }
20  }
```

Fonte: *print* de autoria própria.

6. MONGO DB

Para desempenhar o papel do banco de dados de todo o sistema, utilizou-se o *Mongo DB*. Para tanto, instalou-se sua extensão no *Visual Studio Code*.

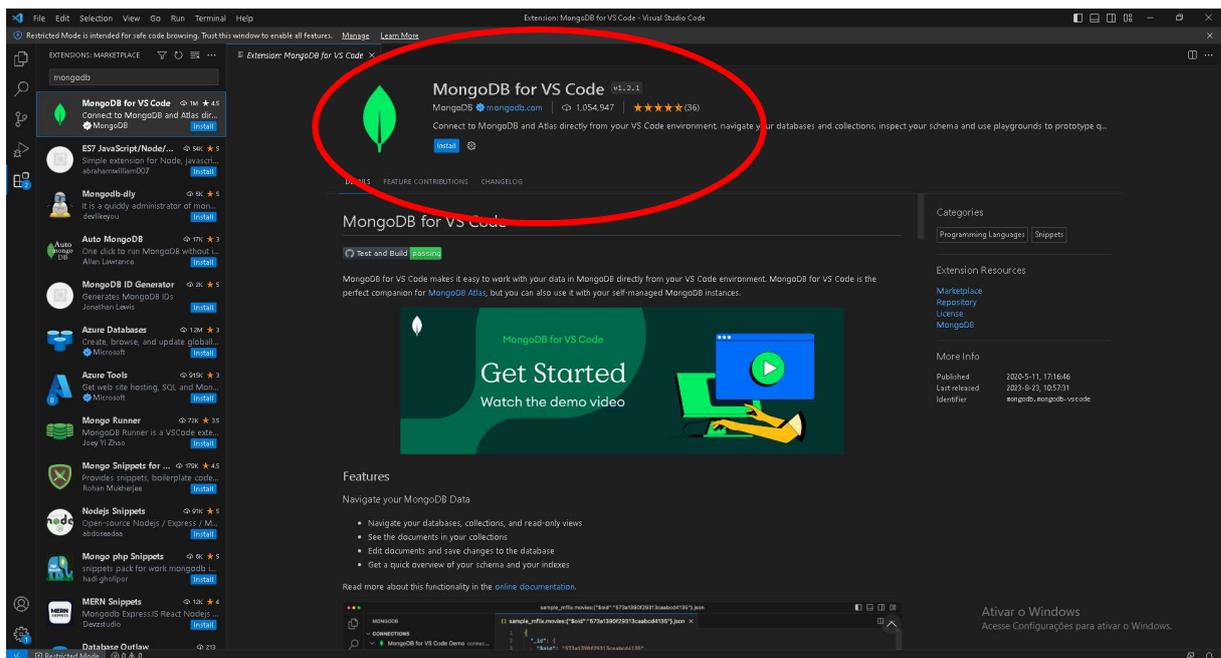
Figura 13. Logotipo do Mongo DB.



Fonte:

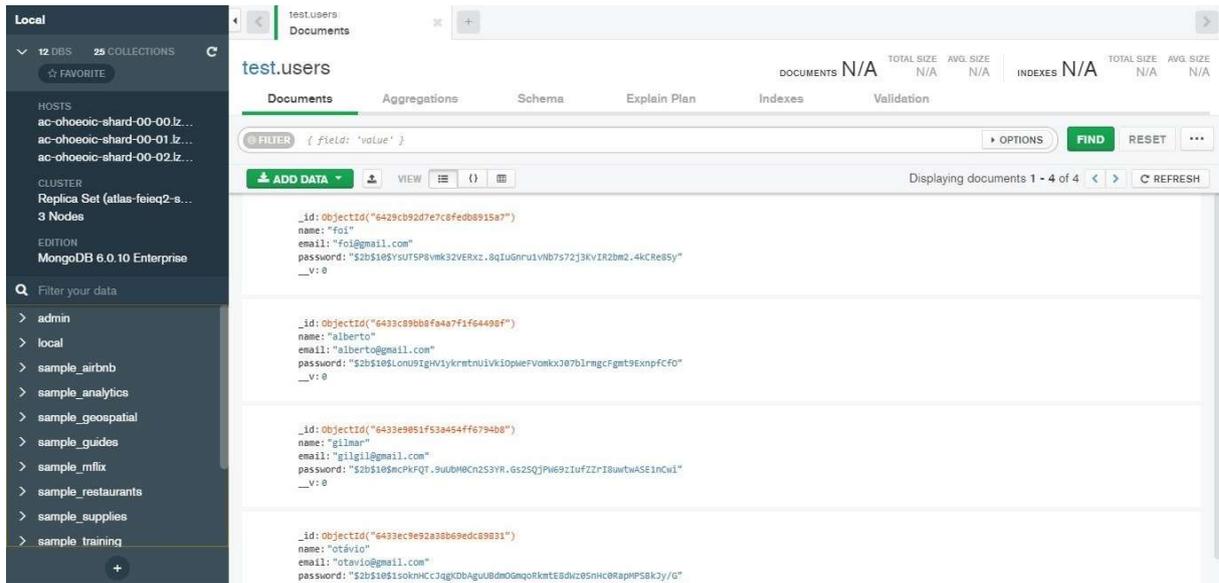
https://pt.m.wikipedia.org/wiki/Ficheiro:MongoDB_Logo.svg

Figura 14. Print indicando a extensão que deve ser instalada.



Fonte: *print* de autoria própria.

Figura 15. Print de parte do banco de dados do sistema, construído por meio do *Mongo DB*.



Fonte: print de autoria própria.

7. TENSORFLOW.JS

Para a tarefa de reconhecimento e classificação de imagens, escolheu-se o *Tensorflow.js*, uma biblioteca de aprendizado de máquina em *JavaScript*. Para isso, a instalação de suas extensões no *Visual Studio Code* foi fundamental.

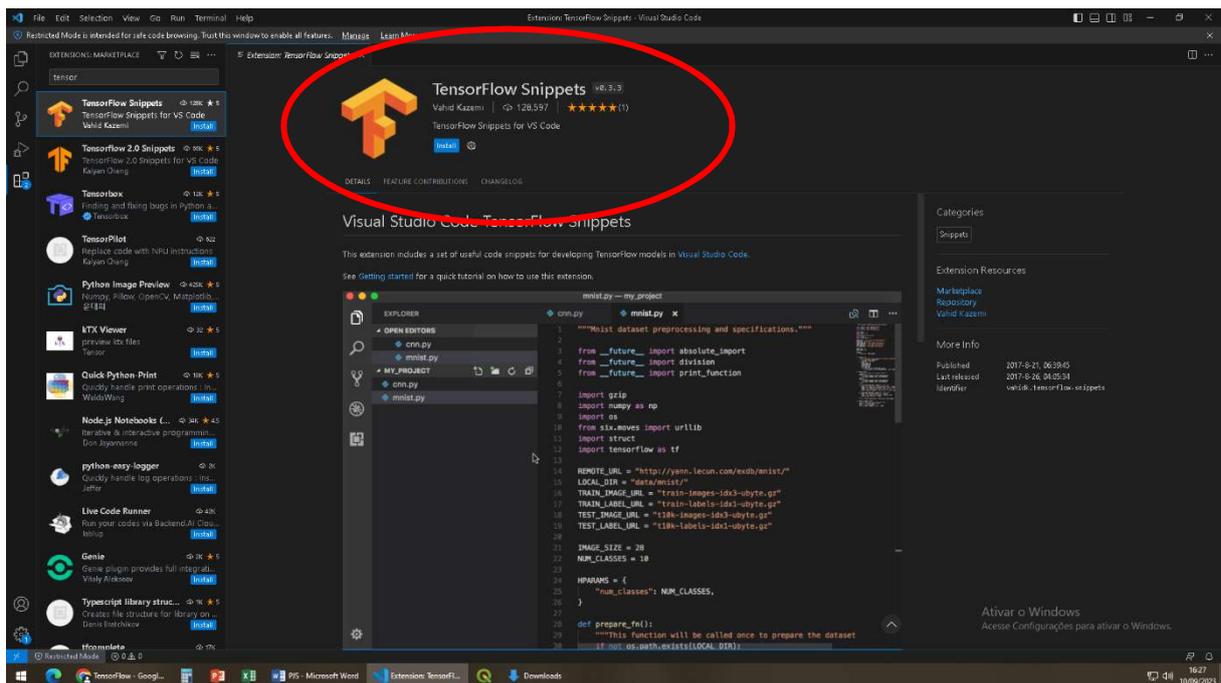
Figura 16. Logotipo do *TensorFlow.js*.



Fonte:

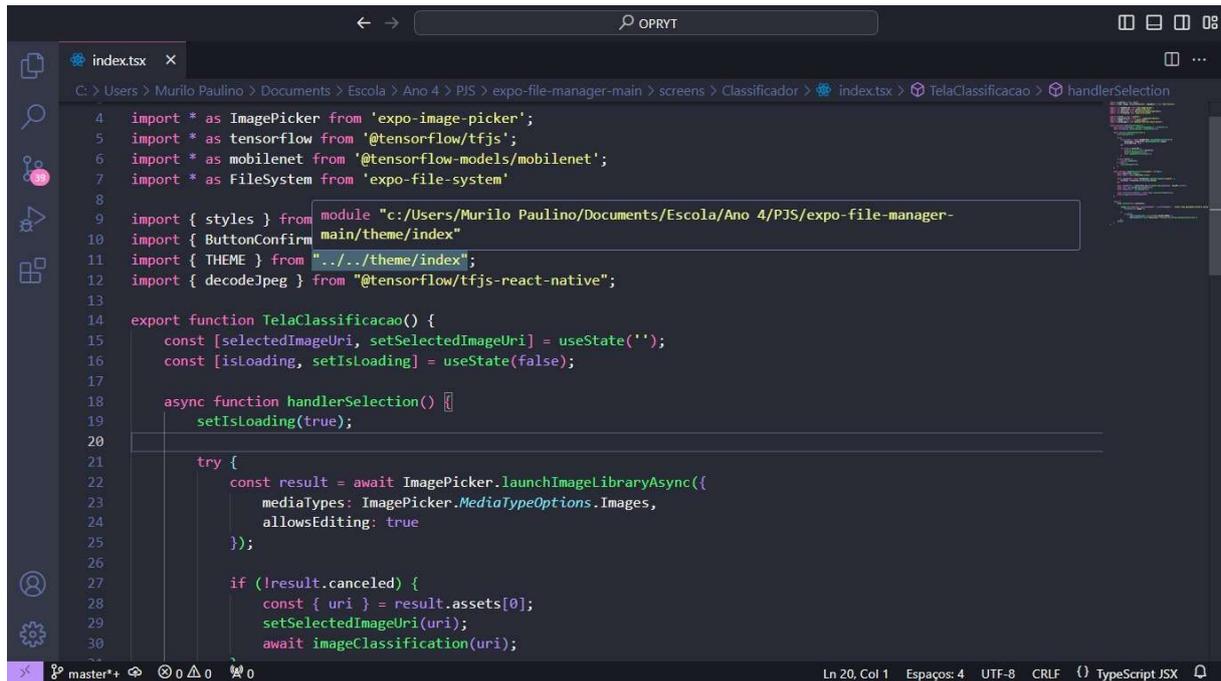
<https://www.tensorflow.org/?hl=pt-br>

Figura 17. Print indicando a extensão que deve ser instalada.



Fonte: print de autoria própria.

Figura 18. Print de código escrito em TensorFlow.js, TypeScript e React Native.



```
index.tsx x
C:\Users\Murilo Paulino\Documents\Escola\Ano 4\PJS\expo-file-manager-main\screens\Classificador > index.tsx > TelaClassificacao > handlerSelection
4 import * as ImagePicker from 'expo-image-picker';
5 import * as tensorflow from '@tensorflow/tfjs';
6 import * as mobilenet from '@tensorflow-models/mobilenet';
7 import * as FileSystem from 'expo-file-system'
8
9 import { styles } from module "c:/Users/Murilo Paulino/Documents/Escola/Ano 4/PJS/expo-file-manager-
10 import { ButtonConfirm main/theme/index"
11 import { THEME } from "../././theme/index";
12 import { decodeJpeg } from "@tensorflow/tfjs-react-native";
13
14 export function TelaClassificacao() {
15   const [selectedImageUri, setSelectedImageUri] = useState('');
16   const [isLoading, setIsLoading] = useState(false);
17
18   async function handlerSelection() {
19     setIsLoading(true);
20
21     try {
22       const result = await ImagePicker.launchImageLibraryAsync({
23         mediaTypes: ImagePicker.MediaTypeOptions.Images,
24         allowsEditing: true
25       });
26
27       if (!result.canceled) {
28         const { uri } = result.assets[0];
29         setSelectedImageUri(uri);
30         await imageClassification(uri);
31       }
32     }
33   }
34 }
```

Ln 20, Col 1 Espaços: 4 UTF-8 CRLF TypeScript JSX

Fonte: *print* de autoria própria.

8. VISUALIZAÇÃO DO APLICATIVO

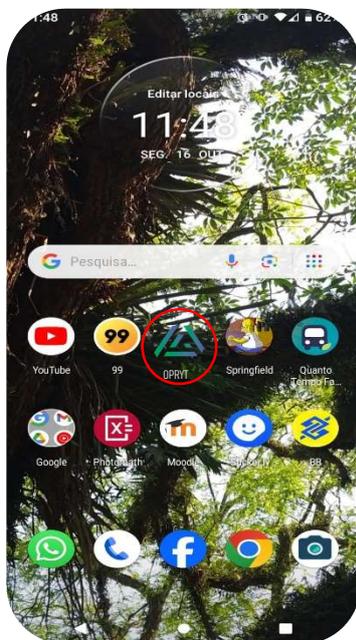
Ao clicar no ícone do aplicativo elaborado (em destaque na figura 20), o usuário é direcionado para a tela de carregamento para a iniciação do sistema (figuras 21 e 22). Após o carregamento ser concluído, o usuário será encaminhado para a tela de *login* (figuras 23 e 24). Nela, o utilizador optará por *logar no software* por meio de seu cadastro já realizado no aplicativo, informando seu *e-mail* e senha cadastrados, ou com alguma conta *Google* existente.

Figura 19. Logotipo do aplicativo desenvolvido.



Fonte: confecção própria dos autores.

Figura 20. Print de tela do menu de um *smartphone* destacando o ícone do aplicativo desenvolvido.



Fonte: confecção própria dos autores.

Figura 21. Tela de carregamento de iniciação do sistema no tema claro.



Fonte: confecção própria dos autores.

Figura 22. Tela de carregamento de iniciação do sistema no tema escuro.



Fonte: confecção própria dos autores.

Figura 23. Tela de *login* do aplicativo no tema claro.



Fonte: confecção própria dos autores.

Figura 24. Tela de *login* do aplicativo no tema escuro.



Fonte: confecção própria dos autores.

Caso o usuário não deseje ou não possua uma conta vinculada à rede da *Google* e também não possua um registro criado no aplicativo, ele poderá criar um clicando em “Registrar-se” na tela exposta por meio das figuras 23 e 24 e, após isso, efetuar o cadastro proposto (figuras 25 e 26), onde a pessoa deverá fornecer ao sistema um *e-mail*, seu nome e uma senha que servirá de acesso ao aplicativo.

Além disso, há a opção “Pular”, caso o usuário não deseje realizar nenhuma forma de login. No entanto, utilizando esta opção, nenhuma garantia de que todos os processos fiquem gravados no dispositivo do usuário é fornecida, além da impossibilidade acessar todos os processos realizados em outros dispositivos.

Figura 25. Tela de cadastro no aplicativo no tema claro.



Fonte: confecção própria dos autores.

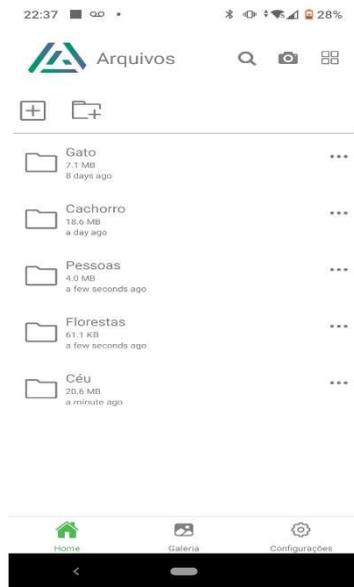
Figura 26. Tela de cadastro no aplicativo no tema escuro.



Fonte: confecção própria dos autores.

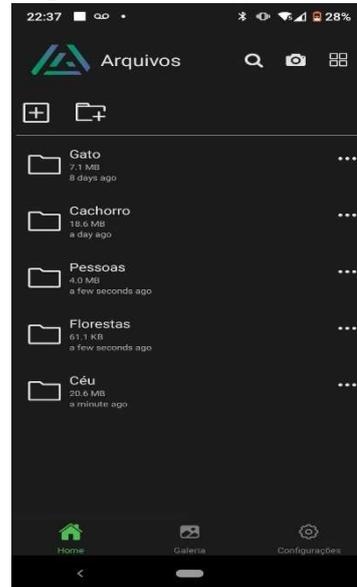
Feito o *login*, o utente será direcionado para o menu central do aplicativo, onde estão dispostas todas as pastas de categorizações criadas, isso se o usuário já tiver utilizado as funcionalidades do aplicativo anteriormente (figuras 27 e 28). Se nenhuma função do aplicativo for utilizada, não haverá nenhuma pasta criada.

Figura 27. Tela do menu central do aplicativo no tema claro.



Fonte: confecção própria dos autores.

Figura 28. Tela do menu central do aplicativo no tema escuro.



Fonte: confecção própria dos autores.

Feito isso, o processo de organização (automático ou não) de imagens pode ser realizado ao importar mídias da galeria do *smartphone*.

Figura 29. Tela da seleção de mídias no tema claro.



Fonte: confecção própria dos autores.

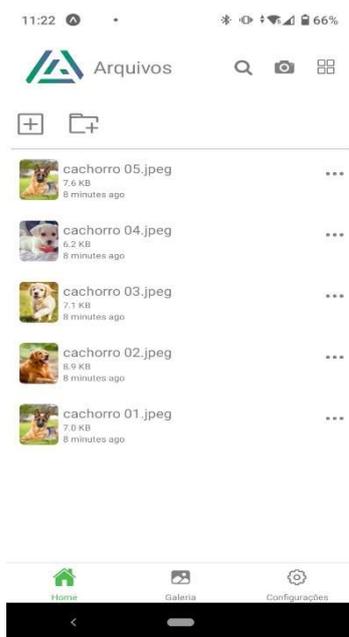
Figura 30. Tela da seleção de mídias no tema escuro.



Fonte: confecção própria dos autores.

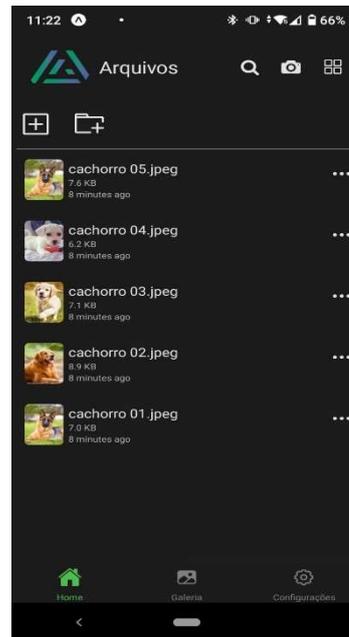
Feito isso, o processamento e tratamento das imagens é feito pelo *software*, armazenando-as nas pastas designadas para cada uma. Um exemplo é demonstrado nas figuras a seguir com fotografias de cães.

Figura 31. Imagens armazenadas após o processamento no tema claro.



Fonte: confecção própria dos autores.

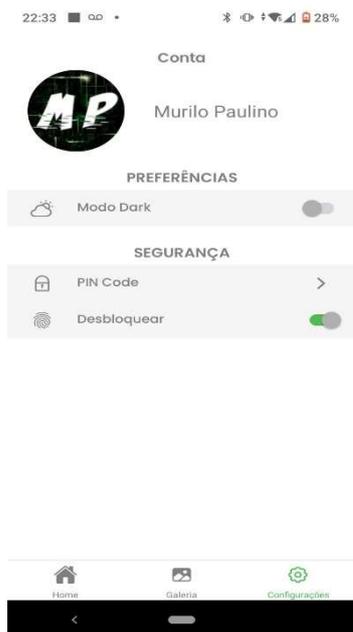
Figura 32. Imagens armazenadas após o processamento no tema escuro.



Fonte: confecção própria dos autores.

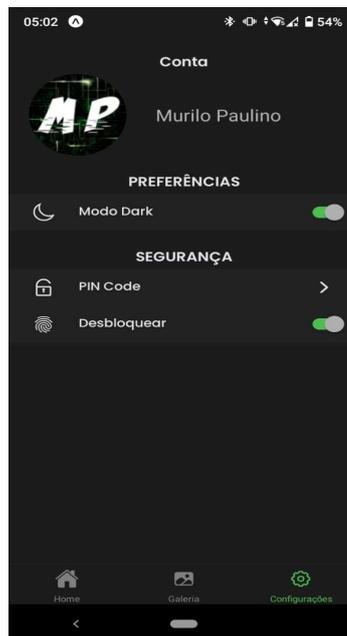
Além disso, há configurações de estilo e segurança. Em relação ao estilo, o usuário pode optar em utilizar o tema claro ou escuro do *software*. Já no tocante à segurança, o usuário pode optar por restringir o acesso ao aplicativo por meio de senha ou coleta da sua impressão digital.

Figura 33. Tela de configurações no tema claro.



Fonte: confecção própria dos autores.

Figura 34. Tela de configurações no tema escuro.



Fonte: confecção própria dos autores.

8.1 Visualização da página web

Além do aplicativo, foi desenvolvida uma página dinâmica *web*. Nesse ambiente, o usuário pode efetuar seu cadastro (caso não tenha uma conta em nosso banco de dados) ou *login* com o propósito de fazer o *download* do aplicativo mobile em seu dispositivo.

Figura 35. Tela principal da página web.



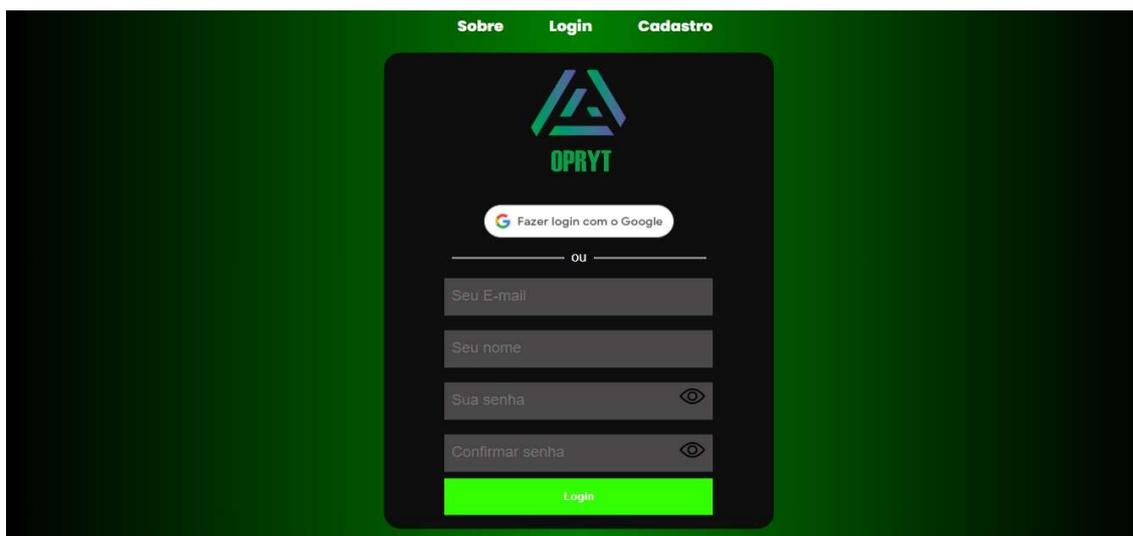
Fonte: confecção própria dos autores.

Figura 36. Tela da seção “sobre” da página web.



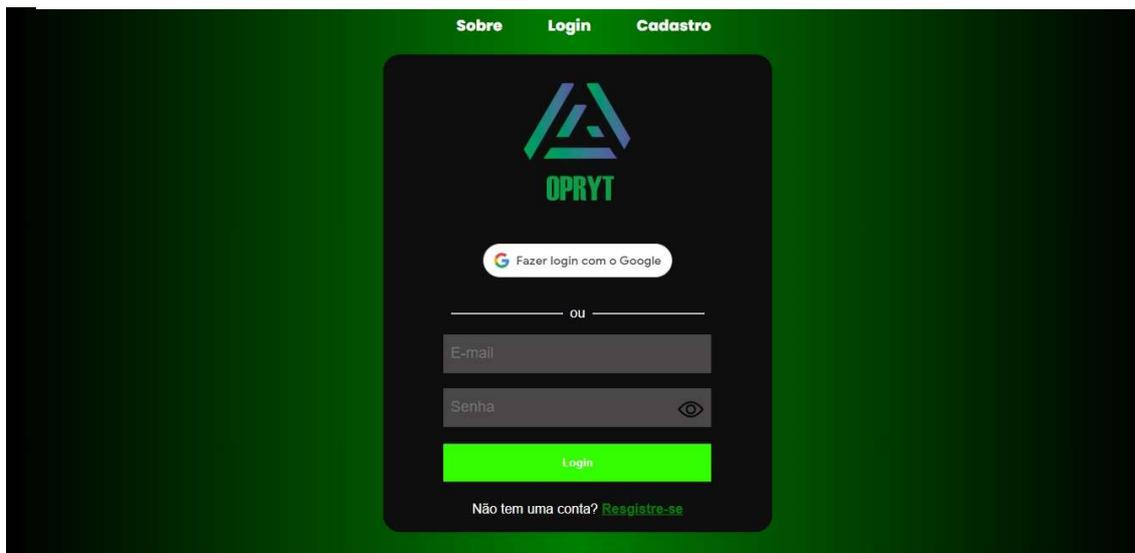
Fonte: confecção própria dos autores.

Figura 37. Tela da seção “cadastro” da página web.



Fonte: confecção própria dos autores.

Figura 38. Tela da seção “login” da página web.



Fonte: confecção própria dos autores.

Dessa forma, o aplicativo desenvolvido está pronto para ser utilizado e exercer sua função: organizar as mídias digitais presentes nas galerias de *smartphones*.

REFERÊNCIAS BIBLIOGRÁFICAS

CUNHA, A. React Native: o que é e tudo sobre o Framework. **Alura**, 2023. Disponível em: <<https://www.alura.com.br/artigos/react-native>>. Acesso em: 08 jul. 2023.

FRANCISCATO, A. Qual a vantagem do MongoDB sobre os outros bancos de dados?. **Dio**, 2023. Disponível em: <<https://www.dio.me/articles/qual-a-vantagem-do-mongodb-sobre-os-outros-bancos-de-dados>>. Acesso em: 08 jul. 2023.

NOÇÕES básicas de JavaScript. **Mdn Web Docs**, [s.d.]. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Learn/Getting_started_with_the_web/JavaScript_basics>. Acesso em: 08 jul. 2023.

O TensorFlow.js é uma biblioteca para machine learning no JavaScript. TensorFlow [s.d.]. Disponível em: <<https://www.tensorflow.org/js?hl=pt-br>>. Acesso em: 08 jul. 2023.

TROQUATTE, D. Curso de Javascript, Typescript e Nodejs. **Vida FullStack**, [s.d.]. Disponível em: <<https://vidafullstack.com.br/curso-de-javascript-typescript-e-nodejs/>>. Acesso em: 08 jul. 2023.