UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS
INE-DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

PEDRO HENRIQUE TAGLIALENHA (22203674)
RITA LOURO BARBOSA (22203157)
VITOR PRAXEDES CALEGARI (22200379)

HACKATON: ConfirmaID

FLORIANÓPOLIS
2024

Resumo

Este trabalho apresenta o desenvolvimento do aplicativo ConfirmaID, destinado a combater fraudes de identidade em interações digitais. Baseado nos conceitos de funções de hash e assinaturas digitais, o sistema verifica a identidade de usuários em processos de cadastro e autenticação, assegurando privacidade e integridade dos dados. A arquitetura do sistema adota uma estrutura modular orientada a funcionalidades, promovendo organização e escalabilidade. Além disso, o trabalho discute a aplicação prática de conceitos teóricos, desafios técnicos enfrentados e a aderência às regulamentações de proteção de dados, destacando o papel do sistema como um modelo mínimo viável (MVP) para soluções mais amplas no contexto de identidade digital.

Palavras-chave: Anti Fraude. Flutter. Hash. Assinatura Digital. Identificação Social.

Sumário

1 INTRODUÇÃO....4
2 FUNDAMENTAÇÃO TEÓRICA....5
2.1 CONFIDENCIALIDADE, INTEGRIDADE E DISPONIBILIDADE.... 5
2.2 AUTENTICIDADE E NÃO REPÚDIO....5
2.3 HASH.... 6
2.4 IDENTIDADE....6
2.5 IDENTIDADE DIGITAL....7
2.6 ROUBO DE IDENTIDADE DIGITAL e GOLPES DIGITAIS....8
2.7 ASSINATURA DIGITAL E ASSINATURA ELETRÔNICA AVANÇADA....8
2.7.1 ASSINATURA DIGITAL....8
2.7.2 ASSINATURA ELETRÔNICA AVANÇADA....11
3 DESCRIÇÃO DO PROJETO....12
4 TRABALHOS CORRELATOS....13
5 ETAPAS DE DESENVOLVIMENTO....14
5.1 ETAPA DE DESIGN.... 14
5.2 ARQUITETURA DE SOFTWARE.... 16
6 DISCUSSÃO....17
6.1 MODELO DE CONFIANÇA NO SISTEMA.... 17
6.2 CADASTRO E VALIDAÇÃO DE IDENTIDADES.... 18
6.3 MITIGAÇÃO DE ATAQUES E RESTRIÇÕES DE IMPLEMENTAÇÃO....18
6.4 CONFORMIDADE COM A LEI GERAL DE PROTEÇÃO DE DADOS.... 20
7 EXPERIMENTAÇÃO....21
7.1 CÓDIGO DESENVOLVIDO....21
7.2 FEATURES.... 22
7.2.1 auth_req_screen.... 22
7.2.2 data_manage.... 23
7.2.3 initial_screen.... 24
7.2.4 login....25
7.2.5 profile_search....26
7.2.6 profile_verification_screen....27
7.2.7 results_screens.... 28
7.2.8 tutorial....29
7.3 FLUXO DAS TELAS DESENVOLVIDAS.... 29
7.4 USO DE SISTEMAS EXTERNOS:....35
a) APIs externas:....35
b) Sistema de backend:....35
c) Site hospedado no Github:....35
8 TRABALHO EM GRUPO E DIFICULDADES ENCONTRADAS.... 36
9 CONCLUSÃO....37
10 REFERÊNCIAS.... 38

1 INTRODUÇÃO

A fraude de identidade é um problema crescente em ambientes digitais e físicos, caracterizado pelo uso indevido de informações pessoais para a obtenção de benefícios ou a execução de atividades maliciosas. Este trabalho aborda especificamente fraudes em interações não presenciais, como chamadas telefônicas, comunicações por aplicativos de mensagem e e-mails. Esses cenários apresentam desafios relacionados à verificação da identidade dos interlocutores[1], que muitas vezes exploram vulnerabilidades em sistemas e práticas de autenticação para enganar indivíduos ou organizações.

Este documento apresenta o desenvolvimento do aplicativo ConfirmaID, projetado para combater fraudes de identidade utilizando conceitos aprendidos durante as aulas da disciplina. Desenvolvido em Flutter para Android e iOS, o aplicativo propõe uma solução baseada em Hash e Assinaturas Digitais para verificar a identidade de usuários em interações digitais. A arquitetura do sistema será detalhada, abrangendo as escolhas de implementação, as ferramentas utilizadas e o processo de design.

A escolha deste tema foi motivada pela relevância crescente das fraudes de identidade no contexto atual, impulsionada pela digitalização de processos e interações. A dificuldade em identificar interlocutores de forma confiável em ambientes digitais é uma vulnerabilidade crítica que demanda soluções práticas e seguras. Este trabalho busca contribuir para esse cenário, propondo uma aplicação acessível para os usuários finais.

Serão explorados os fundamentos teóricos das tecnologias utilizadas, incluindo a geração e o uso de hashes para integridade de dados e o emprego de assinaturas digitais para autenticação no sistema. O funcionamento do esquema será ilustrado com exemplos práticos e imagens explicativas. Além disso, serão discutidos potenciais problemas, limitações e possíveis aprimoramentos, a fim de avaliar a viabilidade e a eficácia da solução proposta.

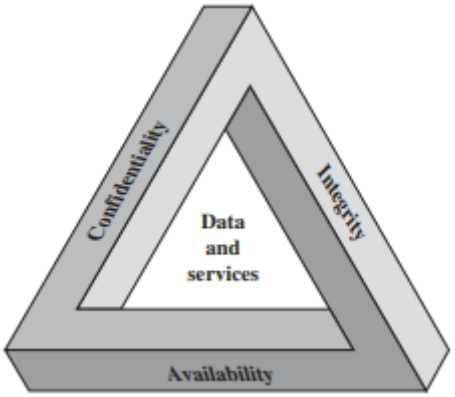
[1] É o termo que se refere a cada uma das pessoas que participam de uma conversa ou de um processo de interação.

2 FUNDAMENTAÇÃO TEÓRICA

Esta seção aborda os conceitos fundamentais para o desenvolvimento do ConfirmaID. São apresentados os princípios de funções hash, assinaturas digitais e fraude de identidade, destacando suas definições, características e relevância na construção de soluções tecnológicas voltadas para a segurança e autenticação em ambientes digitais. Além disso, nesta seção será feita uma delimitação semântica de alguns termos comumente usados no contexto de cibersegurança. Ao fazer uma análise da literatura, o grupo percebeu que os termos podem adorar diferentes sentidos e significados, dependendo do autor e do contexto que foram empregados. Sendo assim, alinhar o entendimento de tais termos no escopo deste trabalho facilitará o entendimento comum do texto.

2.1 CONFIDENCIALIDADE, INTEGRIDADE E DISPONIBILIDADE

Imagem 1 - A Tríade de Requisitos de Segurança



Fonte: [2]

a) **Confidencialidade** [2]:
   i) **Confidencialidade dos dados:** Garante que informações privadas ou confidenciais não sejam disponibilizadas ou divulgadas a indivíduos não autorizados.
   ii) **Privacidade:** Garante que os indivíduos controlem ou influenciem quais informações relacionadas a eles podem ser coletadas e armazenadas e por quem e para quem essas informações podem ser divulgadas.

a) **Integridade**: Integridade significa que os dados são confiáveis, completos e não foram alterados ou modificados acidentalmente por um usuário não autorizado. [3]

b) **Disponibilidade:** Garantir acesso oportuno e confiável e uso de informações. [2]

A aplicação dessa tríade como base de direcionamento no propósito e no desenvolvimento deste projeto será melhor explicitada ao longo do relatório.

2.2 AUTENTICIDADE E NÃO REPÚDIO

a) **Autenticidade:** A propriedade de ser genuíno e poder ser verificado e confiável; confiança na validade de uma transmissão, mensagem ou originador de mensagem. Isso significa garantir que os usuários são quem dizem ser e se cada entrada que chega ao sistema veio de uma fonte confiável. [2]
b) **Não repúdio:** Fornece proteção contra a negação por uma das entidades envolvidas em uma comunicação de ter participado de toda ou parte da comunicação [2] .

2.3 HASH

Uma função *hash* H é uma função que recebe um bloco de dados de comprimento variável 'M' como entrada e produz um valor 'h' chamado *'hash-value'* ou '*hash'* de tamanho fixo, em que h = H(M) . Uma função hash considerada "boa" é aquela que, ao ser aplicada sob grande conjunto de entradas, produzirá resultados uniformemente distribuídos e aparentemente aleatórios (equiprobabilidade dos resultados). Alem disso, em funções de hash a mudança de qualquer bit ou conjunto de bits em M resultará, com grande probabilidade, numa mudança no hash da saída. [2]. Essas características são reflexo de 3 importantes propriedades das funções de *hash*:

- ***preimage resistance (one-way function)***: Para essencialmente qualquer saída pré especificada, é computacionalmente inviável encontrar qualquer entrada que gere este hash em específico. Ou seja, é computacionalmente inviável encontrar qualquer pré imagem x' tal que h(x') = y diante de qualquer y para qual a entrada correspondente é desconhecida [4]
- ***2nd-preimage resistance (weak collision resistance)*:** É computacionalmente inviável encontrar qualquer segunda entrada que tenha a mesma saída de hash que uma determinada entrada escolhida. Ou seja, dada uma determinada entrada x, é computacionalmente inviável encontrar a segunda pré imagem x' != x tal que h(x) = h(x') [4]
- ***Collision resistance (strong collision resistance):*** É computacionalmente inviável encontrar quaisquer 2 entradas distintas x e x' para as quais o hash seja idêntico, ou seja, h(x) = h(x'). (Note que, nesta propriedade, ambas as entradas são desconhecidas e arbitrárias) [4]

Para este trabalho, o hash foi utilizado como mecanismo de garantia de confidencialidade dos dados de contato dos indivíduos cadastrados no sistema. As propriedades acima citadas garantem o segredo dos dados até mesmo para os desenvolvedores com acesso pleno ao banco de dados, já que todas as informações sensíveis são armazenadas apenas com seu hash. Além disso, o hash está presente implicitamente no processo de criação das assinaturas digitais que serão utilizadas em nosso sistema de cadastro.

2.4 IDENTIDADE

O propósito deste projeto tem íntima ligação com a segurança da identidade digital do indivíduo. Para facilitar a compreensão dessa ligação e melhor delimitar o escopo de nossa atuação, será necessário adotar, de antemão, algumas definições de termos base atrelados à identidade digital.

- **Identificador:** Um identificador distingue uma pessoa, lugar ou coisa distinta dentro do contexto. Cada identificador é significativo apenas no contexto para o qual foi criado e apenas quando associado à coisa que está sendo identificada. Portanto, cada identificador pode ser razoavelmente pensado como tendo um conjunto <coisa identificada, identificador, namespace>, por exemplo,em nosso banco de dados relacional, <Pessoa, número de registro único, banco de dados do confirma ID>. [4]
- **Atributo:** Um atributo é uma característica associada a uma entidade, como um indivíduo. Exemplos de atributos persistentes incluem cor dos olhos, data de nascimento, CPF e nome completo. Exemplos de atributos temporários incluem endereço, empregador, função organizacional, e, como é nosso foco, número de telefone, contas em redes sociais e demais formas de contato digital. [5]
- **Identificador pessoal**: Identificadores persistentes consistem naquele conjunto de identificadores associados a um indivíduo que são **baseados em atributos que são difíceis ou impossíveis de alterar.** Por exemplo, data de nascimento e padrão genético são todos identificadores pessoais. Observe que qualquer um pode mentir sobre sua data de nascimento, mas ninguém pode alterá-la. [5]
- **Identificação:** Identificação é a associação de um **identificador pessoal** com um **indivíduo apresentando atributos**, por exemplo, "Você é Rita Louro Barbosa". Exemplos incluem aceitar a associação entre uma pessoa física e um nome reivindicado, ou CPF; [4]
- **Autenticação**: Autenticação é prova de um atributo. Por exemplo, apresentar um documento de identidade oficial e original com foto a um fiscal é uma forma de autenticação do nome que você declara possuir. [5]
- **Autenticação de identidade.** Autenticação de identidade é provar uma associação entre uma entidade (um indivíduo, por exemplo)e um identificador: a associação de uma pessoa com um histórico de crédito ou educacional; a associação de um automóvel com uma placa de licença; ou uma pessoa com uma conta bancária. Essencialmente, isso é a verificação da reivindicação <coisa identificada, coisa> em um contexto. Observe que "Você é Rita Louro Barbosa." é identificação, enquanto "Seus documentos ilustram que você é Rita Louro Barbosa." é autenticação de identidade. [5]
- **Autenticação de Atributos.** A autenticação de um atributo é provar uma associação entre uma entidade e um atributo; Em um sistema de identidade, esse é geralmente um processo de duas etapas: autenticação de identidade seguida pela autenticação da associação do atributo e do identificador. Observe a diferença entre "Seus documentos ilustram que você é Rita Louro Barbosa" e "Seus documentos ilustram que você é um aluno registrado na UFSC e tem direitos de acesso neste prédio". [5]
- **Autorização.** Autorização é uma decisão para permitir uma ação específica com base em um identificador ou atributo. Exemplos incluem a capacidade de uma pessoa fazer reivindicações em linhas de crédito; o direito de um veículo de emergência passar por um sinal vermelho; ou uma certificação de um dispositivo resistente à radiação a ser conectado a um satélite em construção. [5]

- **Identidade**.Uma identidade é uma representação de uma entidade em um domínio de aplicação específico. Uma identidade é formada por um conjunto de identificadores referentes à entidade. [6]

2.5 IDENTIDADE DIGITAL

Uma identidade digital é um relacionamento um-para-um entre uma entidade com existência concreta (Humano, empresa, organização) e sua presença digital. Uma presença digital pode consistir em várias contas, credenciais e direitos associados a um indivíduo. [7] Ou seja, é a identidade de uma entidade (indivíduo ou empresa, por exemplo) no domínio digital.

É importante aqui diferenciar identidade digital de perfis e contas, por exemplo. Um usuário da internet pode ter múltiplos perfis ou múltiplas contas em uma mesma plataforma. Estas contas podem ser anônimas (não estarem explicitamente ligadas à identidade digital do indivíduo), e podem também, por outro lado, serem identificadas com atributos e identificadores que as relacionem com a identidade digital deste indivíduo.

2.6 ROUBO DE IDENTIDADE DIGITAL e GOLPES DIGITAIS

Roubo de identidade, um subconjunto de fraude de identidade, refere-se especificamente ao ato de roubar informações pessoais ou financeiras de alguém para cometer fraude.[8].

Dentre os crimes mais comuns associados ao roubo de identidade, destacam-se:

- **Phishing:** utiliza comunicações falsas, como e-mails ou mensagens de texto, que aparentam ser de fontes confiáveis, como pessoas conhecidas, bancos, instituições governamentais ou empresas. O objetivo é induzir as vítimas a fornecer informações sensíveis, como senhas, números de cartão de crédito ou CPF. Essas informações podem ser usadas pelos criminosos para acessar contas financeiras ou realizar compras fraudulentas. [9]
- **Perfis falsos e catfishing:** Criminosos criam **perfis falsos** em redes sociais ou aplicativos de mensagens, muitas vezes usando fotos e informações roubadas de outras pessoas. Esses perfis podem ser utilizados para enganar terceiros, estabelecer relacionamentos de confiança ou até aplicar golpes financeiros, como o pedido de transferências monetárias sob falsas alegações. [10]
- **Roubo de contas:** A partir do acesso não autorizado a e-mails, redes sociais ou outros serviços digitais, os criminosos podem se passar pela vítima, enviar mensagens fraudulentas a contatos ou utilizar a conta para disseminar golpes. [11]

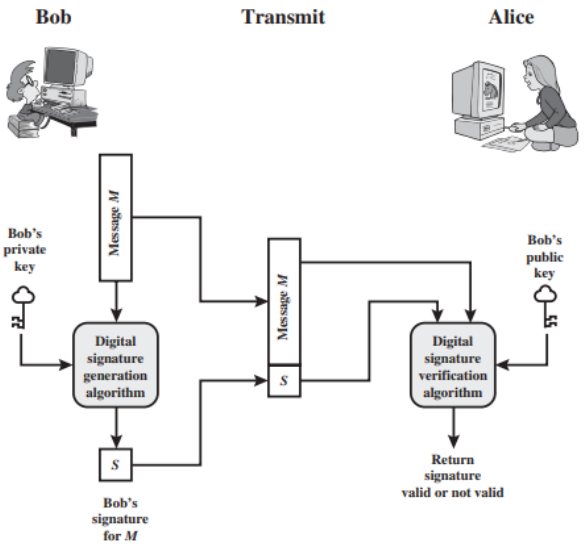
Nestes crimes, identificadores pessoais, como fotos faciais e nome completo, e atributos temporários atrelados à identidade são apropriados pelos criminosos e utilizados para fazer as vítimas acreditarem que estão em contato com determinada pessoa ou empresa, quando, na verdade, estão em contato com os golpistas.

2.7 ASSINATURA DIGITAL E ASSINATURA ELETRÔNICA AVANÇADA

2.7.1 ASSINATURA DIGITAL

Uma assinatura digital é um mecanismo de autenticação que permite ao criador de uma mensagem anexar um código que atua como uma assinatura. Normalmente, a assinatura é formada pegando o hash da mensagem e criptografando a mensagem com a chave privada do criador. A assinatura garante a origem e a integridade da mensagem. [2]

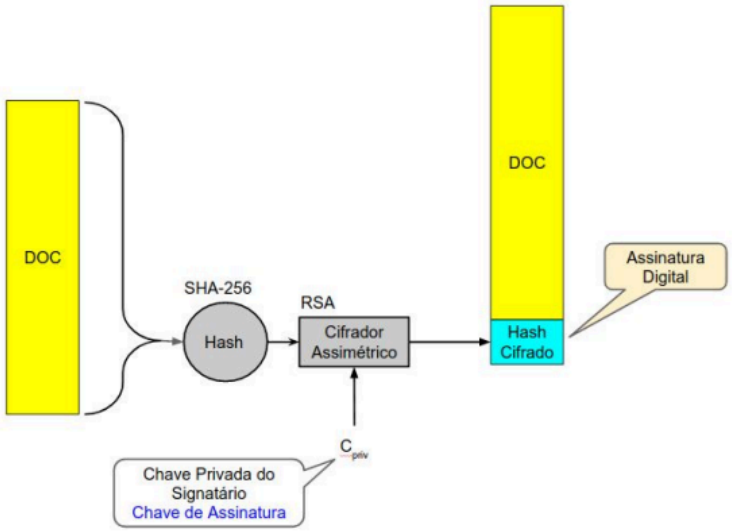
Imagem 2 - Modelo Genérico do Processo de Assinatura Digital.



Fonte:[2]

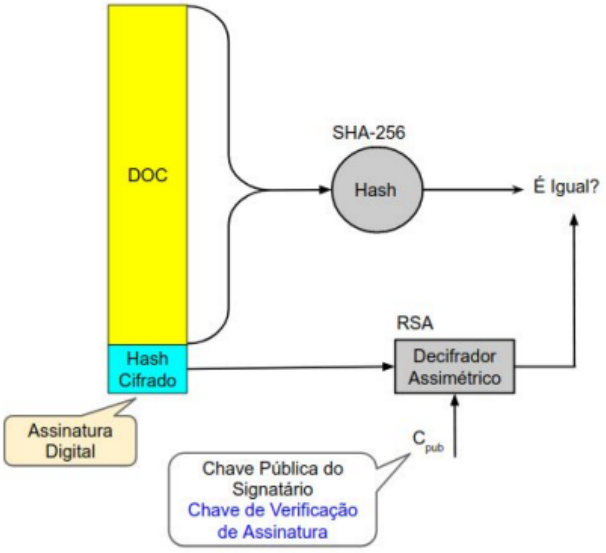
Uma relevante forma de aplicação das assinaturas digitais é no desenvolvimento das assinaturas eletrônicas com propriedades de autenticidade, integridade e não repúdio, asseguradas por alguma cadeia de confiança estabelecida. Essas assinaturas tem propósito similar àquelas escritas em papel: assegurar a identidade do indivíduo envolvido em alguma ação, contrato ou acordo. Elas são desenvolvidas utilizando processos criptográficos que se baseiam no uso de função de hash e cifradores RSA (como ilustrado na imagem 2), que permitem a posterior verificação da validade da assinatura (imagens 3 e 4).

Imagem 3 - Assinatura Digital de Documentos Eletrônicos



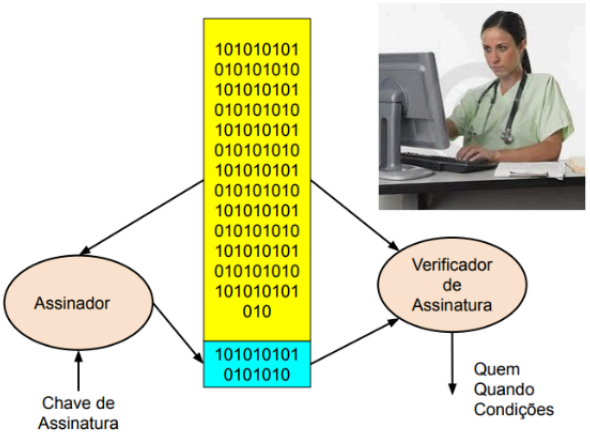
Fonte: [12]

Imagem 4 - Verificação da Assinatura



Fonte: [12]

Imagem 5 - Ilustração didática da verificação da Assinatura



Fonte: [12]

O uso de assinaturas digitais no contexto de assinatura eletrônica é válida e reconhecida legalmente no Brasil[2]. Em termos legais, no Brasil, há 3 classificações de níveis de assinatura eletrônica, para os casos de interação com os entes de governo: Assinatura eletrônica simples, avançada e qualificada [13] . Neste trabalho, daremos enfoque na assinatura avançada:

[2] Leis : Lei nº 14.063, de 23 de Setembro de 2020 , Medida Provisória nº 2.200-2, de 24 de Agosto de 2001 e **DECRETO Nº 10.543, DE 13 DE NOVEMBRO DE 2020**

2.7.2 ASSINATURA ELETRÔNICA AVANÇADA

É a assinatura eletrônica provida por meio do portal Gov.br, a partir da Lei n. 14.063/20, que regula o tipo de assinatura a ser utilizado nas relações entre o cidadão e os serviços públicos. Todo cidadão tem acesso a essa modalidade de assinatura desde que se qualifique ao modelo de autenticação existente no portal. Esse serviço é provido pelo ITI à Plataforma Gov.br gerida pela Secretaria de Governo Digital (SGD) do Ministério da Gestão e da Inovação em Serviços Públicos (MGI).

Esse tipo de assinatura tem validade jurídica conforme a Lei 14.063/2020 e regulamentado pelo Decreto nº 10.543, de 13/11/2020. [14]

Diante dessa validade jurídica estabelecida legalmente no território brasileiro, e assumindo a competência da Secretaria do Governo Federal para o estabelecimento dos mecanismos e etapas de verificações de segurança e identidade no processo de fornecimento do serviço de assinatura eletrônica pelo Gov.br, consideramos que tal assinatura é suficientemente segura para ser adotada como meio de validação da identidade dos usuários cadastrados em nosso sistema.

3 DESCRIÇÃO DO PROJETO

O projeto consiste em um aplicativo voltado para a verificação de identidade digital, projetado para reduzir fraudes relacionadas à manipulação de informações de contato. O sistema permite que os usuários registrem suas informações, como números de telefone, e-mails ou identificadores em plataformas como Discord, vinculando esses dados a suas identidades por meio de assinaturas digitais baseadas no modelo de confiança do Gov.br. A assinatura digital assegura que cada registro seja autenticamente associado à pessoa que o submeteu, garantindo que a pessoa é quem realmente diz ser.

O funcionamento do aplicativo baseia-se na capacidade de pesquisa de dados de contato para verificar sua presença no banco de dados, sem expor as informações armazenadas. Um usuário pode consultar, por exemplo, se um número de telefone ou e-mail associado a um determinado nome está registrado no sistema. A verificação é realizada comparando os hashes gerados pela entrada do usuário com os armazenados no banco de dados. Caso os valores coincidam, o sistema confirma a correspondência, permitindo ao usuário confirmar a autenticidade da informação. Essa abordagem evita que o sistema seja utilizado como uma lista telefônica pública, garantindo a privacidade das informações.

4 TRABALHOS CORRELATOS

Ao pesquisarmos a respeito de sistemas com propostas similares à nossa, não foram encontradas soluções parecidas ou que abordassem a mesma problemática da centralização da verificação dos dados de contato pertencentes a indivíduos e entidades. Entretanto, cabe citar aqui uma solução proposta para o gerenciamento de identidades digitais, com a utilização de uma rede descentralizada baseada em blockchain [15].

5 ETAPAS DE DESENVOLVIMENTO

A seção a seguir descreve as etapas de desenvolvimento do aplicativo, abrangendo desde o design inicial até a definição da arquitetura de software. Serão abordadas as principais decisões tomadas em cada fase, incluindo o uso de ferramentas específicas e abordagens técnicas que garantiram a funcionalidade, segurança e usabilidade do sistema.

5.1 ETAPA DE DESIGN

O desenvolvimento do design do aplicativo foi um processo que envolveu tanto aspectos visuais quanto estruturais, com o objetivo de criar uma solução eficiente que satisfizesse os requerimentos do projeto. As escolhas de design foram orientadas por padrões estabelecidos pelo Padrão Digital de Governo[3], que oferece diretrizes para assegurar a consistência e acessibilidade em sistemas interativos. Essas normas foram seguidas para garantir que o aplicativo ofereça uma experiência de uso intuitiva, em conformidade com as expectativas de aplicações governamentais.

A implementação visual do aplicativo foi realizada utilizando o Adobe XD[4], uma ferramenta voltada para a criação de interfaces e experiências digitais. Nesse ambiente, foram elaborados protótipos de alta fidelidade que definem os aspectos visuais e as rotas entre as telas do aplicativo. Esses protótipos consideraram elementos como tipografia, espaçamentos, botões e curvas, de modo a facilitar o desenvolvimento e garantir a padronização entre os diferentes desenvolvedores. Além disso, as decisões de design para manter boas práticas em UI (Interface do Usuário) e UX (Experiência do Usuário) utilizou como referência o TCC de Victor Hardt [1], que apresentou uma análise de usabilidade do aplicativo Medida Inteligente, aplicativo onde integrantes da equipe tinham trabalhado anteriormente. Essas telas estão exemplificadas nas Figuras 5 e 6 abaixo.

A identidade visual do aplicativo foi construída com um propósito funcional, utilizando uma paleta de cores selecionadas baseadas em sua acessibilidade e significados. A cor branca predomina para transmitir transparência e clareza na interface, o laranja é utilizado para destacar ações importantes e assegurar acessibilidade, e o cinza reforça a sensação de segurança e confiabilidade.

Outro aspecto crucial no design foi a elaboração de fluxos intuitivos, com telas bem divididas e autoexplicativas. Para usuários que acessam o aplicativo pela primeira vez, foram projetadas telas de tutorial que orientam sobre o funcionamento das principais funcionalidades. A apresentação de resultados utiliza cores vibrantes e símbolos claros para comunicar de forma visual e rápida o sucesso ou a falha na verificação de identidade.

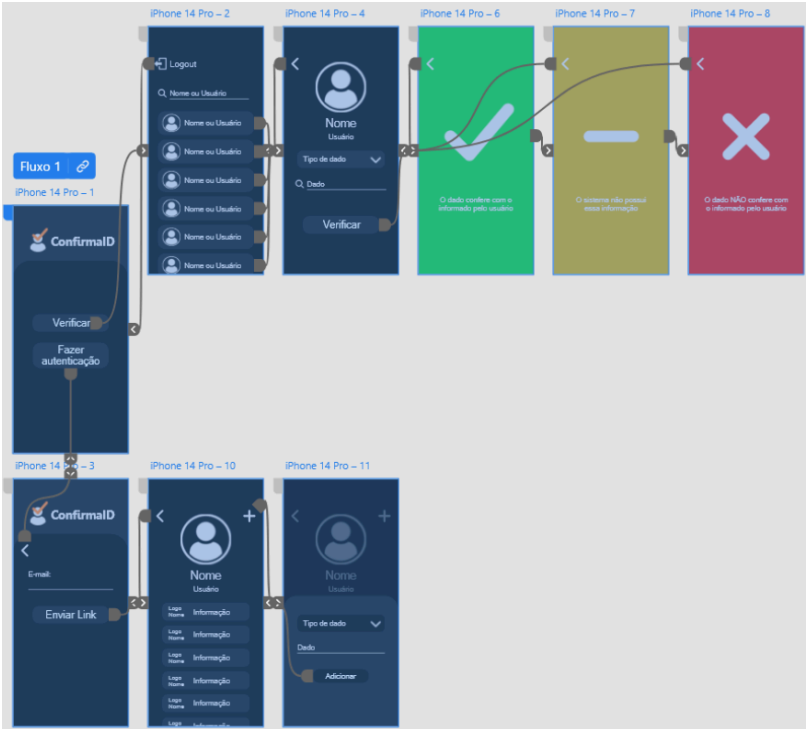
A estrutura inicial do projeto contemplava nove telas principais, abrangendo funcionalidades como autenticação, gerenciamento de dados, busca de perfis e exibição de resultados. Contudo, durante o desenvolvimento, o número de telas aumentou para acomodar a adição de novas funcionalidades e atender a requisitos técnicos específicos.

---

[3] Padrão Digital de Governo - Design System

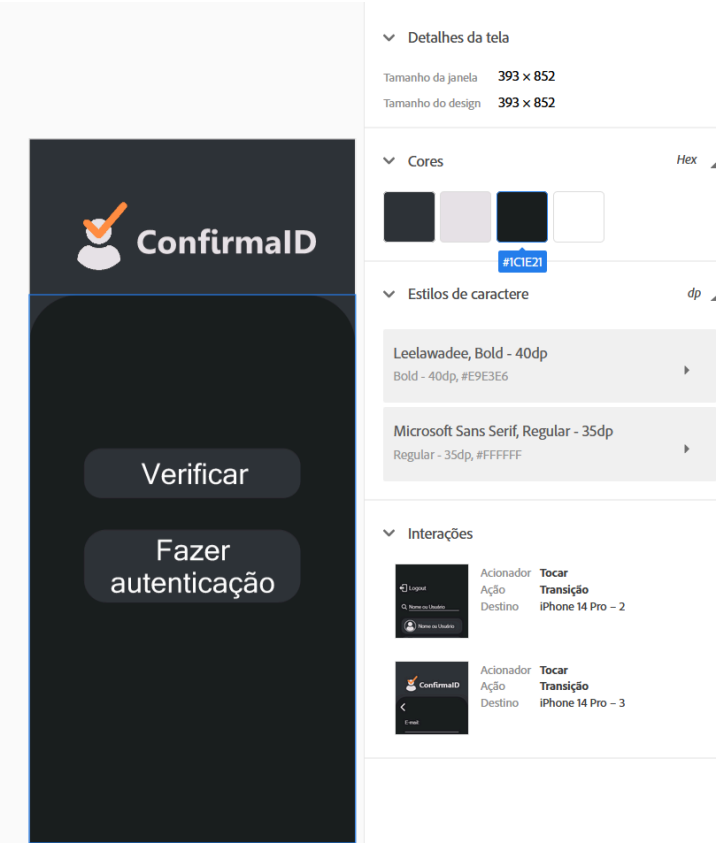
[4] https://xd.adobe.com/view/082f95ac-e56b-48fe-8f47-64a9a7233a27-5d7c/

Figura 5 - Fluxo da telas



Fonte: Elaborado pelos autores(2024).

Figura 6 - Exemplo de tela no Adobe XD



Fonte: Elaborado pelos autores(2024).

5.2 ARQUITETURA DE SOFTWARE

No que diz respeito à arquitetura de software, foi adotado o padrão Feature First Structure[5]. Essa abordagem organiza o código com base nas funcionalidades principais do aplicativo, separando cada funcionalidade em componentes internos específicos. Essa arquitetura foi escolhida por sua capacidade de oferecer organização lógica, simplicidade na manutenção e suporte a ciclos rápidos de desenvolvimento. A estrutura facilita tanto a implementação de novas funcionalidades quanto a correção de problemas.

[5] Flutter Professional Folder Structure: Feature-first or Layer-first? | Coding with T

6 DISCUSSÃO

Nesta seção é abordado os principais desafios e decisões técnicas relacionadas ao desenvolvimento do aplicativo, os modelos de confiança adotados, mecanismos de segurança implementados e limitações identificadas no projeto. Além disso, são exploradas alternativas para aprimoramento futuro, considerando as restrições de prazo, recursos disponíveis e a necessidade de conformidade com padrões de segurança e privacidade, como a Lei Geral de Proteção de Dados.

6.1 MODELO DE CONFIANÇA NO SISTEMA

Um modelo de confiança é o conjunto de estruturas, políticas e entidades que permitem estabelecer a segurança em sistemas de autenticação, validação ou comunicação. Na prática, um modelo de confiança pode ser implementado utilizando uma infraestrutura de chave pública (PKI), na qual autoridades certificadoras (CAs) desempenham um papel central, ou mesmo por meio de sistemas governamentais, como o Gov.br. Esses modelos são projetados para garantir a autenticidade, a integridade e a confiabilidade das transações digitais, permitindo que partes envolvidas confiem umas nas outras com base em mecanismos previamente auditados e estabelecidos. O modelo de confiança é essencial para criar uma cadeia lógica de verificação, que viabiliza a validação de identidades, certificados ou assinaturas digitais com base em uma arquitetura confiável.

Ao optar por utilizar um modelo de confiança já consolidado, como uma PKI governamental, aproveitam-se sistemas que passaram por rigorosa auditoria e são amplamente reconhecidos pela sua segurança e eficácia. Isso reduz significativamente os esforços técnicos e os custos associados ao desenvolvimento e à manutenção de uma infraestrutura própria. A criação de um modelo de confiança independente exigiria não apenas mecanismos equivalentes de segurança e credibilidade, mas também a tarefa desafiadora de conquistar a confiança dos usuários e entidades externas. Este esforço adicional, além de ultrapassar o escopo do trabalho, teria custos desproporcionais em relação aos benefícios, especialmente quando comparado à robustez e à eficiência oferecidas pelos modelos já existentes, como o Gov.br.

Além do modelo de confiança hierárquico tradicional, baseado em uma infraestrutura de chave pública (PKI) com autoridades certificadoras (CAs), existem outras abordagens que poderiam ser consideradas para o aplicativo, como o modelo de confiança descentralizado conhecido como *Web of Trust*. Nesse modelo, a confiança não é estabelecida por uma única entidade central, mas por uma rede distribuída de usuários que validam as chaves uns dos outros, formando uma estrutura de confiança colaborativa[6]. Embora esse modelo ofereça maior resiliência contra falhas ou comprometimentos de uma entidade única, ele apresenta desafios significativos em termos de escalabilidade e usabilidade, especialmente em aplicações com muitos usuários desconhecidos entre si[7]. Outra possibilidade seria o modelo híbrido, que combina elementos de confiança centralizada e descentralizada, como ocorre em algumas implementações de

[6] Web of trust - Wikipedia

[7] What is Web of Trust? - GeeksforGeeks

blockchain, onde as transações são verificadas coletivamente, mas podem se ancorar em entidades confiáveis para aspectos específicos.

6.2 CADASTRO E VALIDAÇÃO DE IDENTIDADES

O principal desafio enfrentado no desenvolvimento do aplicativo foi garantir que a identidade de um usuário fosse validada de forma confiável no momento do cadastro. O uso de assinaturas digitais baseadas no Gov.br introduz a **aplicação prática de um modelo de confiança reconhecida nacionalmente**, conforme discutido nas aulas sobre infraestrutura de chaves públicas (PKI). Este mecanismo possibilita a autenticação ao vincular a identidade do usuário a um certificado digital emitido por uma entidade confiável. A solução desenvolvida solicita que o usuário assine digitalmente um documento no momento do cadastro, utilizando seu certificado do Gov.br. Posteriormente, a assinatura é validada por meio de uma API[8], que verifica sua conformidade e extrai os dados do signatário, como nome completo e os seis primeiros dígitos do CPF, para cruzamento com as informações fornecidas. Essa abordagem possibilita a aplicação dos conceitos de **assinaturas digitais**, **certificados digitais** e **âncoras de confiança** abordados na disciplina.

A opção por não integrar diretamente o sistema ao login do *Gov.br* foi motivada pela complexidade técnica e pelos prazos restritos, mas isso também criou a oportunidade de implementar um mecanismo que destaca conceitos como assinaturas digitais e verificação de conformidade. A alternativa de exigir fotos de documentos e validações manuais, embora simples, adicionaria vulnerabilidades como manipulação de imagens e inconsistências no processo de revisão humana. Em contrapartida, a assinatura digital, vinculada à PKI, assegura integridade e confidencialidade, além de ser escalável para ambientes com grande volume de usuários.

Um problema identificado na versão atual do MVP é a ausência de mecanismos de **Autenticação de Atributos** que confirmem se os canais de comunicação declarados pelo usuário, como números de telefone, e-mails ou contas de redes sociais, realmente pertencem a ele. Atualmente, o sistema aceita essas informações sem realizar verificações adicionais, como o envio de um código de verificação por SMS, e-mail, ou o uso de bots para validação em plataformas como Discord ou Instagram. Essa limitação deixa o aplicativo mais suscetível a possíveis falsificações dessas informações. No futuro poderia ser implementada essa funcionalidade mas, para o desenvolvimento do MVP, foi considerado como não crítico, considerando as limitações de tempo na realização do projeto. Portanto, atualmente a segurança deste ponto em nosso sistema se baseia no fato de que seria pouco útil para o usuário fornecer informações falsas para seus dados de contato, uma vez que estes não ficam explícitos nem expostos. Para reduzir erros acidentais, como digitação incorreta, a solução viável implementada é a exibição de um pop up solicitando ao usuário a confirmação de que o dado inserido está correto antes de enviá-lo ao nosso banco de dados.

6.3 MITIGAÇÃO DE ATAQUES E RESTRIÇÕES DE IMPLEMENTAÇÃO

[8] Documentação da API do Verificador de Conformidade ICP-Brasil

Um possível ataque identificado seria o de repetição, no qual um usuário mal-intencionado poderia obter acesso a um documento previamente assinado por outra pessoa e utilizá-lo para se registrar no aplicativo, tentando se passar por ela. Esse cenário exploraria a reutilização de uma assinatura digital legítima fora do contexto original para fins de fraude. Entretanto, este cenário específico foi considerado de baixo risco, uma vez que o sistema realiza uma verificação prévia para confirmar se o usuário já está cadastrado. A situação plausível seria a de um indivíduo solicitar a remoção de seus dados do sistema e, posteriormente, ter seu documento assinado vazado e utilizado por um terceiro para se registrar. Ainda assim, a improbabilidade desse caso não justifica o custo adicional de implementação das medidas de segurança mencionadas, considerando o escopo e os objetivos do projeto.Para mitigar esse tipo de ataque, poderiam ser implementadas soluções como a inclusão de *nonces*[9] ou tokens temporários, garantindo que cada operação fosse única, ou o uso de *timestamps*, que associariam uma janela de validade a cada transação, restringindo o uso da assinatura a períodos específicos.

No armazenamento de dados, foi utilizado funções de hash criptográfico, SHA-256, garantindo que informações sensíveis, como e-mails e números de telefone, não sejam diretamente armazenadas. Isso impede que o sistema seja explorado como uma lista de informações públicas. Mesmo em caso de vazamento de dados, as funções de hash tornam os valores armazenados irreversíveis sem um esforço computacional significativo. Além disso, foi aplicada a técnica de *salting*, que consiste em adicionar uma sequência de bits aleatórios à informação antes de aplicar o hash. O *salting* evita que valores idênticos produzam hashes iguais, dificultando significativamente a correlação entre bases de dados comprometidas. Isso protege o sistema contra ataques em que um adversário, ao obter um hash vazado em outro contexto, poderia compará-lo com os valores armazenados e inferir informações.

Outras opções para o armazenamento de dados, como criptografia simétrica ou assimétrica, embora úteis em cenários específicos, apresentariam desvantagens neste contexto. A criptografia simétrica exigiria o gerenciamento seguro de chaves, aumentando o risco de comprometimento em larga escala. Por outro lado, a criptografia assimétrica teria um custo computacional elevado para o volume de verificações previstas. A abordagem baseada em hash, ao invés disso, atende aos requisitos de privacidade com simplicidade e eficiência, evitando riscos associados a acesso direto aos dados armazenados.

Adversários potenciais incluem usuários mal-intencionados que buscam comprometer o sistema para roubo de identidade, manipulação de dados ou ataques de negação de serviço (DoS). Os vetores de ataque mais prováveis incluem a exploração de vulnerabilidades no processo de verificação de assinaturas digitais, ataques de repetição ao tentar reutilizar assinaturas válidas e tentativas de acesso não autorizado ao banco de dados. Vetores como ataques de força bruta contra os hashes ainda representam riscos residuais. Para mitigar a vulnerabilidade de ataque de força bruta na verificação de Hash, foi adicionado um timeout de 5 segundos entre consultas, com esse timeout para testar todas as possibilidades de número de telefone de um dado DDD, levaria no mínimo $(((((10^8 \times 5) \div 60) \div 60) \div 24) \div 365) \simeq 15$ anos.

Outro ponto identificado como uma potencial fragilidade na segurança do sistema é a obsolescência dos dados a longo prazo. Isso ocorre quando usuários cadastrados deixam de

[9] What Is a Cryptographic Nonce? Definition & Meaning | Okta

atualizar suas informações de contato em caso de mudanças, resultando em respostas desatualizadas durante consultas no sistema (falsos negativos ou falsos positivos). Esse cenário pode comprometer a confiança do público na precisão do sistema. Para mitigar esse risco, implementamos a exibição da data de cadastro dos dados de contato nas telas de "dado correto" e "dado incorreto". Dessa forma, o usuário pode avaliar há quanto tempo a informação foi atualizada, estimando a probabilidade de obsolescência. Além disso, para futuras implementações, consideramos a inclusão de notificações periódicas solicitando a atualização dos dados. Essas solicitações seriam enviadas em intervalos regulares, incentivando os usuários a manter suas informações sempre atualizadas e garantindo que nosso banco de dados permaneça confiável.

Uma estratégia que poderia melhorar a segurança do sistema seria a implementação de um monitoramento ativo e contínuo. Esse monitoramento incluiria a análise contínua de logs de acesso e operações realizadas no sistema, a detecção de padrões anômalos que possam indicar atividades maliciosas, como tentativas de força bruta ou acesso repetido a recursos sensíveis, e a verificação de integridade dos componentes críticos do sistema, como APIs externas e bases de dados. Alertas automatizados poderiam notificar administradores em tempo real sobre eventos suspeitos, permitindo respostas rápidas para mitigar ataques em andamento, permitindo uma detecção rápida de um ataque e permitindo à equipe começar a contenção antes de maiores estragos. No desenvolvimento do aplicativo, no entanto, a prioridade foi direcionada para desenvolver contramedidas contra os ataques mais prováveis dentro do prazo disponível, garantindo um nível aceitável de segurança enquanto se mantém a viabilidade técnica e econômica do projeto.

6.4 CONFORMIDADE COM A LEI GERAL DE PROTEÇÃO DE DADOS

A conformidade com a Lei Geral de Proteção de Dados (LGPD) foi um dos requisitos técnicos do projeto. O aplicativo coleta apenas os dados necessários para sua operação, todos fornecidos voluntariamente pelos usuários, e mantém as informações sensíveis em formato de hash no banco de dados, garantindo que elas não sejam armazenadas de forma acessível. Apenas o nome completo e o CPF são mantidos sem hash, exclusivamente para fins de identificação interna. Antes de qualquer uso do aplicativo, os usuários são apresentados a um *popup* com os termos de uso, detalhando os dados coletados, como são processados e armazenados, e seus direitos como titulares dos dados.

7 EXPERIMENTAÇÃO

7.1 CÓDIGO DESENVOLVIDO

A pasta *lib/*, que é onde está a grande maioria do código desenvolvido para o aplicativo, está organizada em subpastas que segmentam o código de acordo com suas responsabilidades funcionais, conforme a figura 7. As principais subpastas incluem *features/*, *components/shared/*, e *utils/.* A subpasta *features/* contém módulos específicos do aplicativo, cada um representando uma funcionalidade distinta, como *data_manage/*, *auth_req_screen/*, e outras. A subpasta *components/shared/* armazena componentes reutilizáveis utilizados em múltiplas funcionalidades, garantindo consistência e facilitando a manutenção. A subpasta *utils/* contém utilitários e funções auxiliares que suportam as operações comuns utilizadas em diferentes partes do aplicativo.

As dependências do projeto são definidas no arquivo *pubspec.yaml*, localizado na raiz do projeto. Este arquivo especifica todas as bibliotecas externas necessárias para o funcionamento do aplicativo, incluindo flutter, supabase_flutter para integração com o backend, http para requisições HTTP e HTTPS, e shared_preferences para armazenamento local de dados. Dependências de desenvolvimento, como flutter_test e lint, são também declaradas para suportar testes automatizados e análise estática de código.

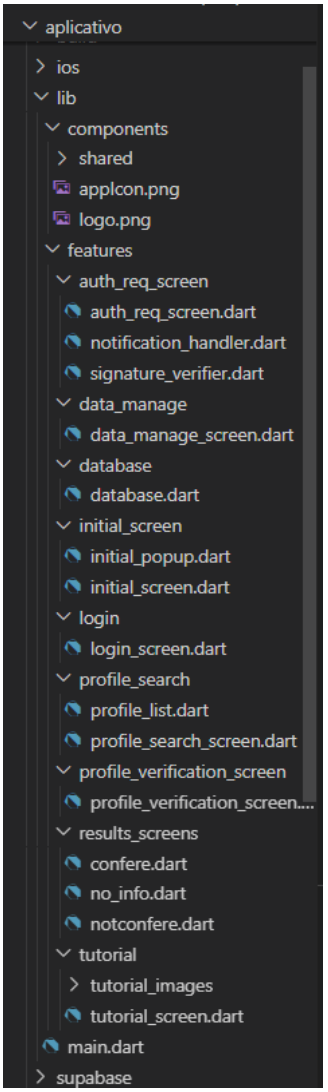
O fluxo da aplicação inicia-se no arquivo *main.dart*, localizado diretamente na pasta *lib/*. Este arquivo configura as variáveis de ambiente, inicializa serviços essenciais como o Supabase, e define o tema e as rotas principais do aplicativo. A navegação entre diferentes telas é gerenciada pelas rotas especificadas nas subpastas de *features/*, utilizando o sistema de rotas do Flutter para transições organizadas entre módulos. O gerenciamento de estado é implementado conforme a necessidade de cada funcionalidade.

As classes principais estão distribuídas dentro das subpastas de *features/*. Por exemplo, na subpasta *data_manage/*, a classe *DataManageScreen* gerencia a interface e a lógica para a atualização dos dados do usuário, enquanto a classe *Database* dentro de *features/database/* gerencia as operações CRUD e a comunicação com o backend Supabase. Na subpasta *auth_req_screen/*, a classe *SignatureVerifier* é responsável pela verificação de assinaturas de declarações. Cada classe é focada em uma única responsabilidade, promovendo a modularidade e facilitando testes unitários.

As features desenvolvidas abrangem funcionalidades como gestão de dados do usuário, verificação de assinaturas, autenticação e autorização, e integração com serviços externos. Por exemplo, a feature de *data_manage* permite aos usuários atualizar suas informações pessoais e realizar operações de leitura e escrita no banco de dados. A feature de *auth_req_screen* implementa a lógica necessária para autenticação de usuários e verificação de segurança através de assinaturas digitais. Cada feature possui sua própria estrutura interna, com subcomponentes e utilitários específicos que suportam suas operações.

Componentes compartilhados na subpasta *components/shared/* incluem widgets de interface de usuário reutilizáveis, como *Avatar*, que exibe imagens de perfil, e outros componentes que são utilizados em múltiplas features para manter a consistência visual e funcional. A subpasta *shared/* contém funções auxiliares para processamento de dados, formatação de informações e outras operações comuns utilizadas em diferentes partes do aplicativo.

Figura 7 - Organização da pasta lib



Fonte: Elaborado pelos autores(2024).

7.2 FEATURES

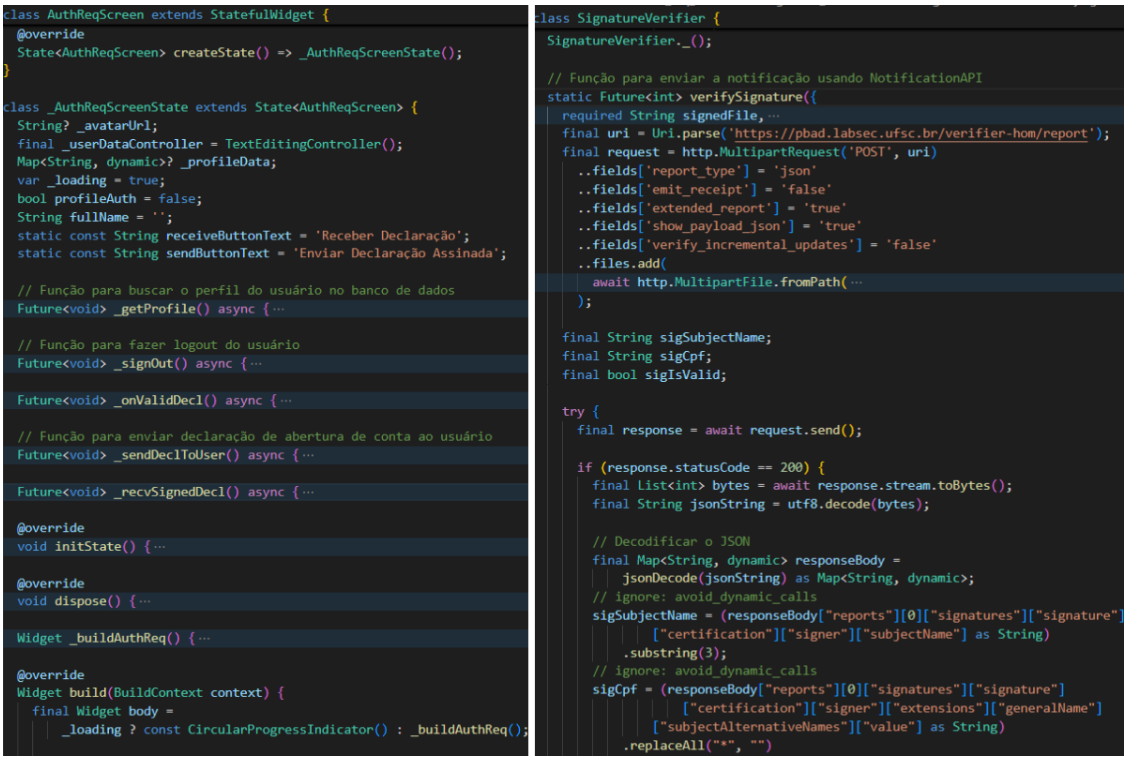
A pasta *features/* é subdividida em módulos que agrupam funcionalidades específicas do aplicativo. Cada subpasta representa uma feature distinta e contém as implementações relacionadas a ela.

7.2.1 auth_req_screen

A pasta *auth_req_screen* contém a implementação da tela de requisição de autenticação. A classe principal é a *AuthReqScreen*, que é um *StatefulWidget* responsável por gerenciar o estado da tela de autenticação. Esta classe lida com a exibição de informações ao usuário, como o envio e recebimento de declarações assinadas. Além disso, a classe utiliza métodos para verificar a assinatura do usuário e atualizar o perfil no banco de dados. A classe *SignatureVerifier* é utilizada para verificar a assinatura digital do usuário, enquanto a classe *NotificationHandler* é responsável por enviar notificações ao usuário, como o email de confirmação de criação de conta.

A classe *AuthReqScreen* utiliza o pacote *connectivity_plus* para verificar a conectividade com a internet antes de realizar operações críticas. A classe também interage com a classe *Database* para buscar e atualizar dados do perfil do usuário. A interface do usuário é construída utilizando widgets do Flutter, como *ElevatedButton*, *TextField* e *ListView*. A classe também implementa métodos para lidar com a seleção de arquivos PDF e a verificação de assinaturas digitais, utilizando a classe *SignatureVerifier*. A classe *NotificationHandler* é utilizada para enviar notificações ao usuário, informando sobre o status das operações de autenticação, ela interage com a API, NotificationAPI, para mandar o email com a declaração .

Figura 8 - Classes auth_req_screen e signature_verifier



```dart
class AuthReqScreen extends StatefulWidget {
  @override
  State<AuthReqScreen> createState() => _AuthReqScreenState();
}

class _AuthReqScreenState extends State<AuthReqScreen> {
  String? _avatarUrl;
  final _userDataController = TextEditingController();
  Map<String, dynamic>? _profileData;
  var _loading = true;
  bool profileAuth = false;
  String fullName = '';
  static const String receiveButtonText = 'Receber Declaração';
  static const String sendButtonText = 'Enviar Declaração Assinada';

  // Função para buscar o perfil do usuário no banco de dados
  Future<void> _getProfile() async {···

  // Função para fazer logout do usuário
  Future<void> _signOut() async {···

  Future<void> _onValidDecl() async {···

  // Função para enviar declaração de abertura de conta ao usuário
  Future<void> _sendDeclToUser() async {···

  Future<void> _recvSignedDecl() async {···

  @override
  void initState() {···

  @override
  void dispose() {···

  Widget _buildAuthReq() {···

  @override
  Widget build(BuildContext context) {
    final Widget body =
        _loading ? const CircularProgressIndicator() : _buildAuthReq();
```

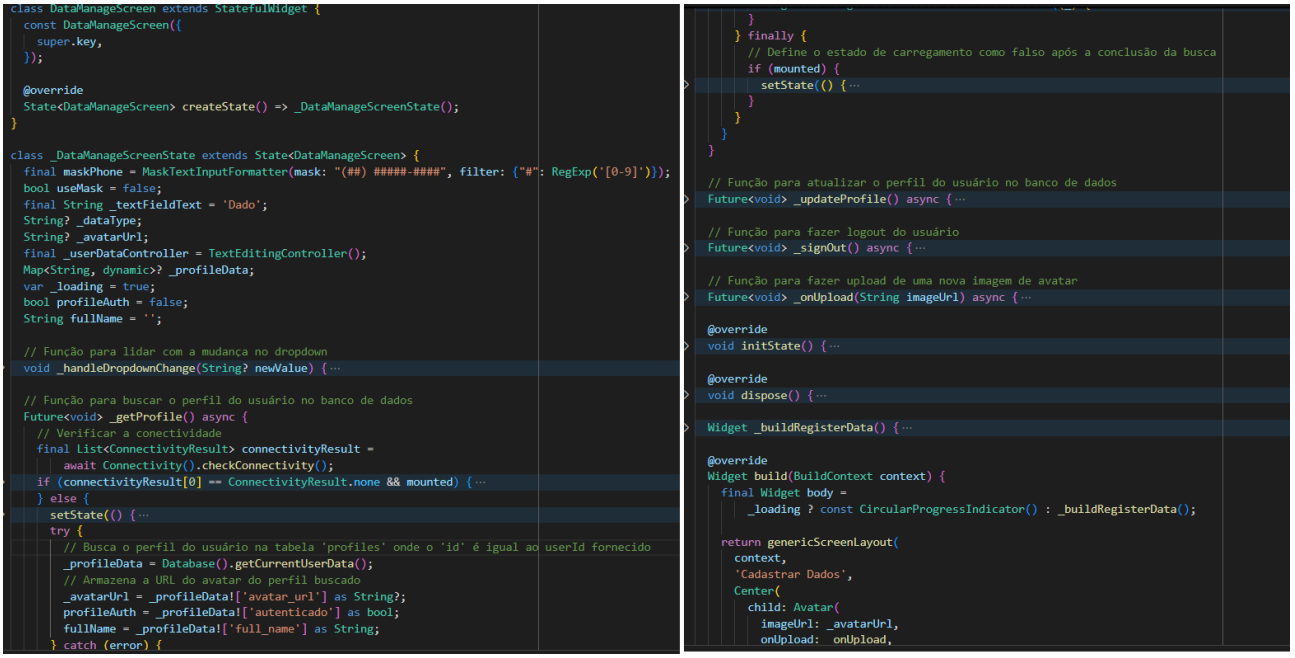
```dart
class SignatureVerifier {
  SignatureVerifier._();

  // Função para enviar a notificação usando NotificationAPI
  static Future<int> verifySignature({
    required String signedFile,···
    final uri = Uri.parse('https://pbad.labsec.ufsc.br/verifier-hom/report');
    final request = http.MultipartRequest('POST', uri)
      ..fields['report_type'] = 'json'
      ..fields['emit_receipt'] = 'false'
      ..fields['extended_report'] = 'true'
      ..fields['show_payload_json'] = 'true'
      ..fields['verify_incremental_updates'] = 'false'
      ..files.add(
        await http.MultipartFile.fromPath(···
      );

    final String sigSubjectName;
    final String sigCpf;
    final bool sigIsValid;

    try {
      final response = await request.send();

      if (response.statusCode == 200) {
        final List<int> bytes = await response.stream.toBytes();
        final String jsonString = utf8.decode(bytes);

        // Decodificar o JSON
        final Map<String, dynamic> responseBody =
            jsonDecode(jsonString) as Map<String, dynamic>;
        // ignore: avoid_dynamic_calls
        sigSubjectName = (responseBody["reports"][0]["signatures"]["signature"]
                ["certification"]["signer"]["subjectName"] as String)
            .substring(3);
        // ignore: avoid_dynamic_calls
        sigCpf = (responseBody["reports"][0]["signatures"]["signature"]
                    ["certification"]["signer"]["extensions"]["generalName"]
                ["subjectAlternativeNames"]["value"] as String)
            .replaceAll("*", "")
```

Fonte: Elaborado pelos autores(2024).

7.2.2 data_manage

A pasta *data_manage* contém a implementação da tela de gerenciamento de dados do usuário. A classe principal é a *DataManageScreen*, que é um *StatefulWidget* responsável por gerenciar o estado da tela de gerenciamento de dados. Esta classe permite ao usuário atualizar informações do perfil, como número de telefone e email, e fazer upload de uma nova imagem de avatar. A classe utiliza métodos para verificar a conectividade com a internet e interage com a classe *Database* para buscar e atualizar dados do perfil do usuário. A classe também implementa métodos para lidar com a seleção de arquivos de imagem e a atualização do perfil do usuário, utilizando a classe *Database*. A classe também utiliza a classe *MaskTextInputFormatter* para formatar a entrada de dados do usuário, como números de telefone.

Figura 9 - DataManageScreen



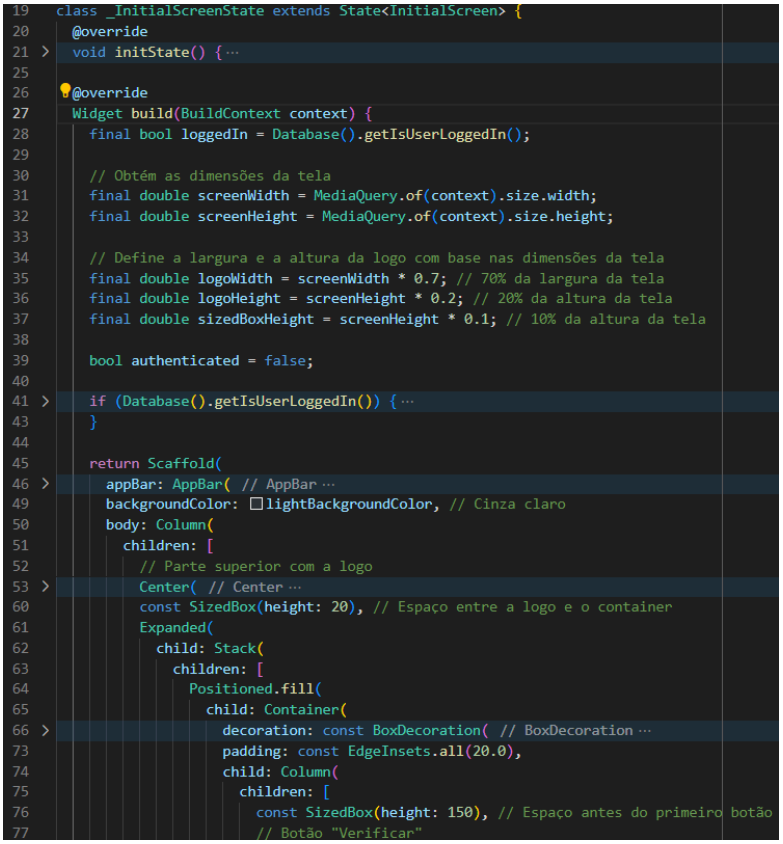
```dart
class DataManageScreen extends StatefulWidget {
  const DataManageScreen({
    super.key,
  });

  @override
  State<DataManageScreen> createState() => _DataManageScreenState();
}

class _DataManageScreenState extends State<DataManageScreen> {
  final maskPhone = MaskTextInputFormatter(mask: "(##) #####-####", filter: {"#": RegExp('[0-9]')});
  bool useMask = false;
  final String _textFieldText = 'Dado';
  String? _dataType;
  String? _avatarUrl;
  final _userDataController = TextEditingController();
  Map<String, dynamic>? _profileData;
  var _loading = true;
  bool profileAuth = false;
  String fullName = '';

  // Função para lidar com a mudança no dropdown
  void _handleDropdownChange(String? newValue) {···

  // Função para buscar o perfil do usuário no banco de dados
  Future<void> _getProfile() async {
    // Verificar a conectividade
    final List<ConnectivityResult> connectivityResult =
        await Connectivity().checkConnectivity();
    if (connectivityResult[0] == ConnectivityResult.none && mounted) {···
    } else {
      setState(() {···
      try {
        // Busca o perfil do usuário na tabela 'profiles' onde o 'id' é igual ao userId fornecido
        _profileData = Database().getCurrentUserData();
        // Armazena a URL do avatar do perfil buscado
        _avatarUrl = _profileData!['avatar_url'] as String?;
        profileAuth = _profileData!['autenticado'] as bool;
        fullName = _profileData!['full_name'] as String;
      } catch (error) {
      }
    } finally {
      // Define o estado de carregamento como falso após a conclusão da busca
      if (mounted) {
        setState(() {···
        }
      }
    }
  }

  // Função para atualizar o perfil do usuário no banco de dados
  Future<void> _updateProfile() async {···

  // Função para fazer logout do usuário
  Future<void> _signOut() async {···

  // Função para fazer upload de uma nova imagem de avatar
  Future<void> _onUpload(String imageUrl) async {···

  @override
  void initState() {···

  @override
  void dispose() {···

  Widget _buildRegisterData() {···

  @override
  Widget build(BuildContext context) {
    final Widget body =
        _loading ? const CircularProgressIndicator() : _buildRegisterData();

    return genericScreenLayout(
      context,
      'Cadastrar Dados',
      Center(
        child: Avatar(
          imageUrl: _avatarUrl,
          onUpload: _onUpload,
```

7.2.3 initial_screen

A pasta *initial_screen* contém a implementação da tela inicial do aplicativo. A classe principal é a *InitialScreen*, que é um *StatefulWidget* responsável por gerenciar o estado da tela inicial. Esta classe exibe a logo do aplicativo e oferece opções para o usuário verificar perfis ou fazer login. A classe utiliza métodos para verificar se o usuário já está logado e, se necessário, exibe um popup com os termos de uso, utilizando a classe *TermsAlertDialog*.

A classe *InitialScreen* utiliza o pacote *shared_preferences* para verificar se o tutorial já foi completado e, se necessário, redireciona o usuário para a tela de tutorial. A classe também implementa métodos para lidar com a navegação entre as telas do aplicativo, utilizando o *Navigator*. A classe *TermsAlertDialog* é utilizada para exibir um popup com os termos de uso, que o usuário deve aceitar antes de continuar a usar o aplicativo.

Figura 10 - initial_screen

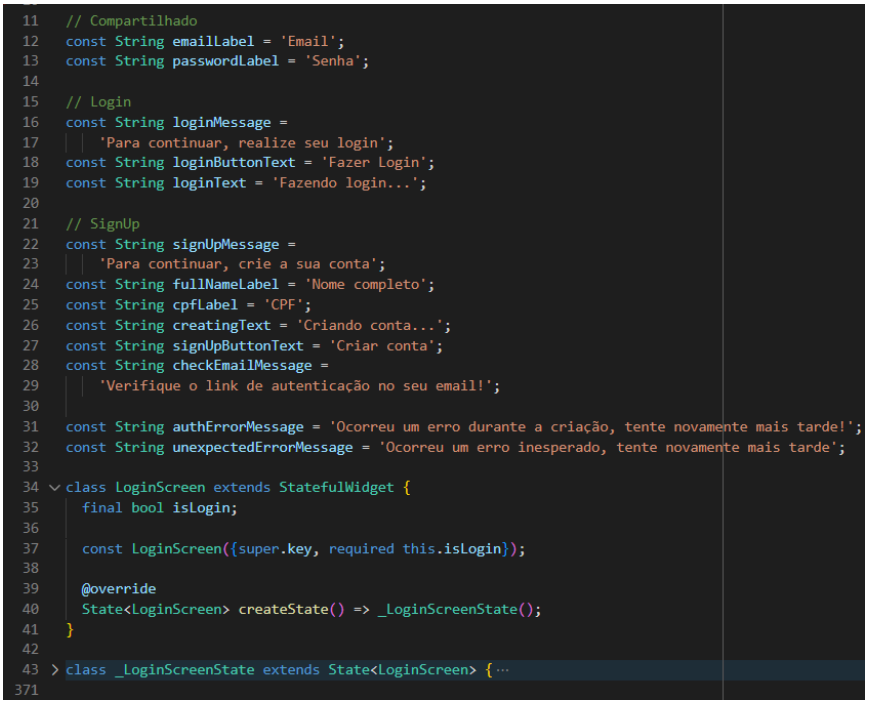


```
class _InitialScreenState extends State<InitialScreen> {
  @override
  void initState() {···

  @override
  Widget build(BuildContext context) {
    final bool loggedIn = Database().getIsUserLoggedIn();

    // Obtém as dimensões da tela
    final double screenWidth = MediaQuery.of(context).size.width;
    final double screenHeight = MediaQuery.of(context).size.height;

    // Define a largura e a altura da logo com base nas dimensões da tela
    final double logoWidth = screenWidth * 0.7; // 70% da largura da tela
    final double logoHeight = screenHeight * 0.2; // 20% da altura da tela
    final double sizedBoxHeight = screenHeight * 0.1; // 10% da altura da tela

    bool authenticated = false;

    if (Database().getIsUserLoggedIn()) {···
    }

    return Scaffold(
      appBar: AppBar( // AppBar ···
      backgroundColor: lightBackgroundColor, // Cinza claro
      body: Column(
        children: [
          // Parte superior com a logo
          Center( // Center ···
          const SizedBox(height: 20), // Espaço entre a logo e o container
          Expanded(
            child: Stack(
              children: [
                Positioned.fill(
                  child: Container(
                    decoration: const BoxDecoration( // BoxDecoration ···
                    padding: const EdgeInsets.all(20.0),
                    child: Column(
                      children: [
                        const SizedBox(height: 150), // Espaço antes do primeiro botão
                        // Botão "Verificar"
```

7.2.4 login

A pasta *login/* contém a implementação da tela de login e cadastro de usuários. A classe principal é a *LoginScreen*, que é um *StatefulWidget* responsável por gerenciar o estado da tela de login e cadastro. Esta classe permite ao usuário fazer login ou criar uma nova conta, dependendo do valor do parâmetro *isLogin*. A classe utiliza métodos para verificar a conectividade com a internet e interage com a classe *Database* para autenticar o usuário e criar novas contas. A classe também implementa métodos para lidar com a validação de CPF e a navegação entre as telas do aplicativo, utilizando o *Navigator*. A classe utiliza a classe *MaskTextInputFormatter* para formatar a entrada de dados do usuário, como CPF e números de telefone.

Figura 11 - login_screen

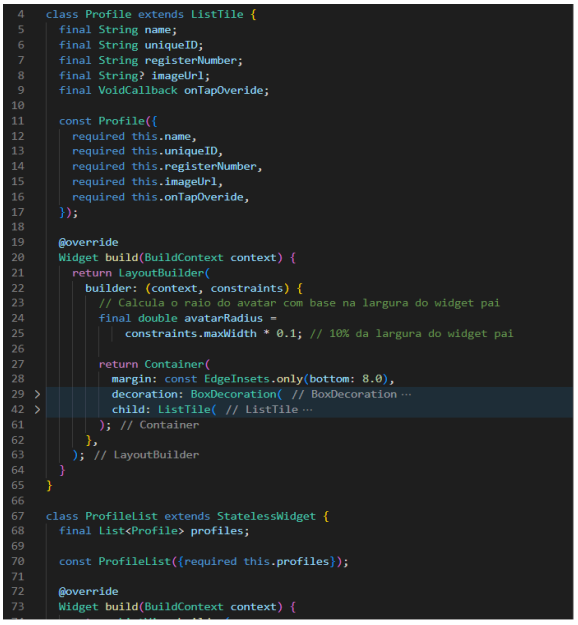


```dart
// Compartilhado
const String emailLabel = 'Email';
const String passwordLabel = 'Senha';

// Login
const String loginMessage =
    'Para continuar, realize seu login';
const String loginButtonText = 'Fazer Login';
const String loginText = 'Fazendo login...';

// SignUp
const String signUpMessage =
    'Para continuar, crie a sua conta';
const String fullNameLabel = 'Nome completo';
const String cpfLabel = 'CPF';
const String creatingText = 'Criando conta...';
const String signUpButtonText = 'Criar conta';
const String checkEmailMessage =
    'Verifique o link de autenticação no seu email!';

const String authErrorMessage = 'Ocorreu um erro durante a criação, tente novamente mais tarde!';
const String unexpectedErrorMessage = 'Ocorreu um erro inesperado, tente novamente mais tarde';

class LoginScreen extends StatefulWidget {
  final bool isLogin;

  const LoginScreen({super.key, required this.isLogin});

  @override
  State<LoginScreen> createState() => _LoginScreenState();
}

class _LoginScreenState extends State<LoginScreen> {···
```

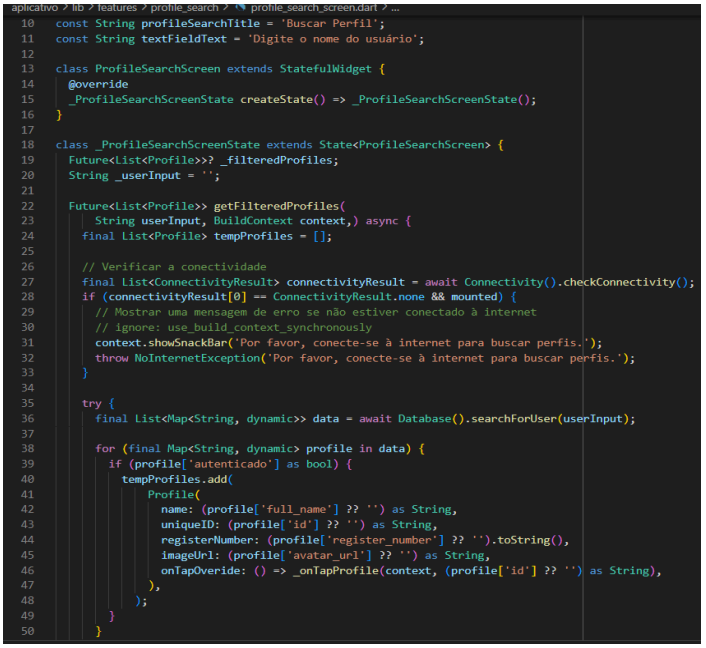
7.2.5 profile_search

A pasta *profile_search* contém a implementação da tela de busca de perfis de usuários. A classe principal é a *ProfileSearchScreen*, que é um *StatefulWidget* responsável por gerenciar o estado da tela de busca de perfis. Esta classe permite ao usuário buscar perfis de outros usuários utilizando um campo de texto. A classe utiliza métodos para verificar a conectividade com a internet e interage com a classe *Database* para buscar perfis no banco de dados. A classe *ProfileSearchScreen* utiliza o pacote *connectivity_plus* para verificar a conectividade com a internet. A classe também implementa métodos para lidar com a exibição dos resultados da busca, utilizando a classe *ProfileList*. A classe *Profile* é utilizada para representar cada perfil de usuário na lista de resultados.

Figura 12 - profile_search_screen e profile_list.



```dart
class Profile extends ListTile {
  final String name;
  final String uniqueID;
  final String registerNumber;
  final String? imageUrl;
  final VoidCallback onTapOveride;

  const Profile({
    required this.name,
    required this.uniqueID,
    required this.registerNumber,
    required this.imageUrl,
    required this.onTapOveride,
  });

  @override
  Widget build(BuildContext context) {
    return LayoutBuilder(
      builder: (context, constraints) {
        // Calcula o raio do avatar com base na largura do widget pai
        final double avatarRadius =
            constraints.maxWidth * 0.1; // 10% da largura do widget pai

        return Container(
          margin: const EdgeInsets.only(bottom: 8.0),
          decoration: BoxDecoration( // BoxDecoration ···
          child: ListTile( // ListTile ···
        ); // Container
      },
    ); // LayoutBuilder
  }
}

class ProfileList extends StatelessWidget {
  final List<Profile> profiles;

  const ProfileList({required this.profiles});

  @override
  Widget build(BuildContext context) {
```



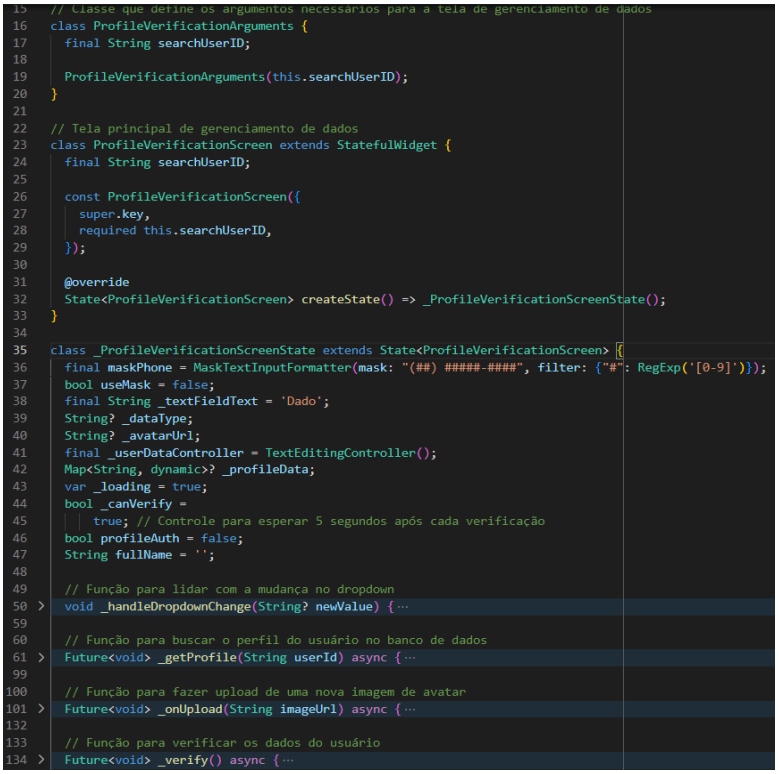
```dart
const String profileSearchTitle = 'Buscar Perfil';
const String textFieldText = 'Digite o nome do usuário';

class ProfileSearchScreen extends StatefulWidget {
  @override
  _ProfileSearchScreenState createState() => _ProfileSearchScreenState();
}

class _ProfileSearchScreenState extends State<ProfileSearchScreen> {
  Future<List<Profile>>? _filteredProfiles;
  String _userInput = '';

  Future<List<Profile>> getFilteredProfiles(
      String userInput, BuildContext context,) async {
    final List<Profile> tempProfiles = [];

    // Verificar a conectividade
    final List<ConnectivityResult> connectivityResult = await Connectivity().checkConnectivity();
    if (connectivityResult[0] == ConnectivityResult.none && mounted) {
      // Mostrar uma mensagem de erro se não estiver conectado à internet
      // ignore: use_build_context_synchronously
      context.showSnackBar('Por favor, conecte-se à internet para buscar perfis.');
      throw NoInternetException('Por favor, conecte-se à internet para buscar perfis.');
    }

    try {
      final List<Map<String, dynamic>> data = await Database().searchForUser(userInput);

      for (final Map<String, dynamic> profile in data) {
        if (profile['autenticado'] as bool) {
          tempProfiles.add(
              Profile(
                name: (profile['full_name'] ?? '') as String,
                uniqueID: (profile['id'] ?? '') as String,
                registerNumber: (profile['register_number'] ?? '').toString(),
                imageUrl: (profile['avatar_url'] ?? '') as String,
                onTapOveride: () => _onTapProfile(context, (profile['id'] ?? '') as String),
              ),
            );
        }
      }
```

7.2.6 profile_verification_screen

A pasta *profile_verification_screen* contém a implementação da tela de verificação de perfis de usuários. A classe principal é a *ProfileVerificationScreen*, que é um *StatefulWidget* responsável por gerenciar o estado da tela de verificação de perfis. Esta classe permite ao usuário verificar se os dados informados conferem com os dados armazenados no banco de dados. A classe utiliza métodos para verificar a conectividade com a internet e interage com a classe *Database* para buscar e verificar dados do perfil do usuário.

A classe *ProfileVerificationScreen* implementa métodos para lidar com a exibição dos resultados da verificação, utilizando as classes *ConfereScreen*, *NotConfereScreen* e *NoInfoScreen*. A classe *ProfileVerificationArguments* é utilizada para passar os argumentos necessários para a tela de verificação de perfis.

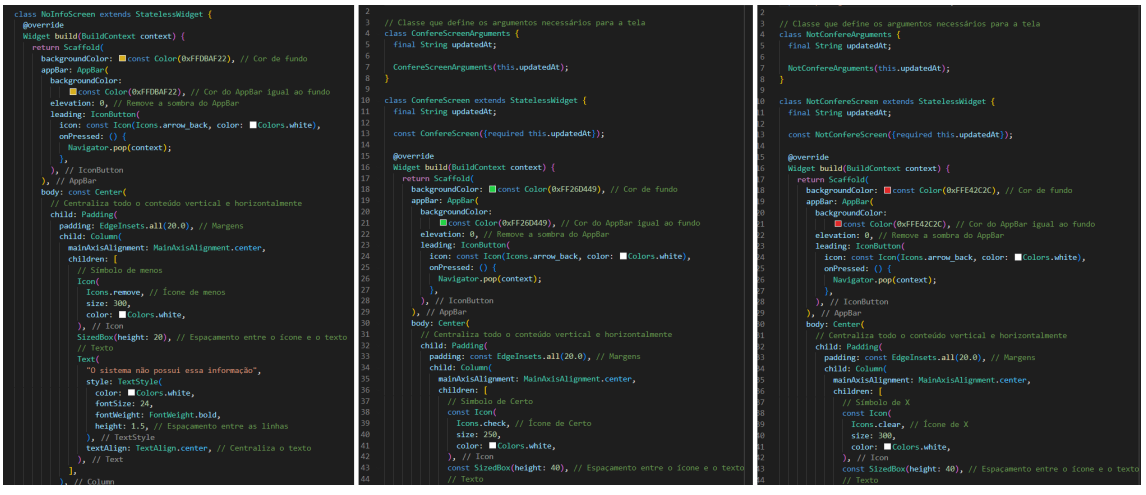
Figura 13 - profile_verification_screen



7.2.7 results_screens

A pasta *results_screens* contém a implementação das telas de resultados da verificação de dados. As classes principais são a *ConfereScreen*, a *NotConfereScreen* e a *NoInfoScreen*, que são StatelessWidget responsáveis por exibir os resultados da verificação de dados. A classe *ConfereScreen* exibe uma mensagem indicando que os dados conferem com os informados pelo usuário, enquanto a classe *NotConfereScreen* exibe uma mensagem indicando que os dados não conferem e a *NoInfoScreen* indica que a informação não consta no Banco de Dados. As classes exibem mensagens e ícones apropriados para indicar o resultado da verificação de dados. Também é mostrado a data de quando foi adicionado a informação no Banco de Dados.

Figura 14- Classes ConfereScreen, NotConfereScreen e NoInfoScreen



Fonte: Elaborado pelos autores(2024).

7.2.8 tutorial

A pasta tutorial contém a implementação da tela de tutorial do aplicativo. A classe principal é a *TutorialScreen*, que é um *StatefulWidget* responsável por gerenciar o estado da tela de tutorial. Esta classe exibe uma série de imagens que guiam o usuário pelos principais recursos do aplicativo. A classe utiliza métodos para salvar a conclusão do tutorial e redirecionar o usuário para a tela inicial após a conclusão. A classe *TutorialScreen* utiliza o pacote *shared_preferences* para salvar a conclusão do tutorial. A classe também implementa métodos para navegar entre as páginas do tutorial e para redirecionar o usuário para a tela inicial após a conclusão do tutorial.

Figura 14- Classes TutorialScreen

```dart
class TutorialScreen extends StatefulWidget {
  @override
  _TutorialScreenState createState() => _TutorialScreenState();
}

class _TutorialScreenState extends State<TutorialScreen> {
  int _currentIndex = 0;
  final int _totalPages = 5;

  // Função para salvar a conclusão do tutorial
  Future<void> _completeTutorial() async {
    final prefs = await SharedPreferences.getInstance();
    await prefs.setBool('tutorialCompleted', true);
  }

  // Função para mudar para a próxima imagem ou finalizar o tutorial
  Future<void> _nextPage() async {
    setState(() {
      if (_currentIndex < _totalPages - 1) {
        _currentIndex++;
      } else {
        _completeTutorial();
        Navigator.pushReplacementNamed(context, '/');
      }
    });
  }

  // Função para mudar para a imagem anterior
  void _previousPage() {
    setState(() {
      if (_currentIndex > 0) {
        _currentIndex--;
      }
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: lightBackgroundColor,
      body: Stack(
        children: [
          if (_currentIndex < _totalPages)
            Positioned.fill(
```

7.3 FLUXO DAS TELAS DESENVOLVIDAS

As imagens da seção 7.3 descrevem o fluxo principal do aplicativo *ConfirmaID*. O aplicativo inicia com telas de tutorial que introduzem os principais recursos e funcionalidades do sistema. O usuário é guiado através de imagens explicativas, que mostram como utilizar o aplicativo para verificar e cadastrar informações. Essas telas têm como objetivo garantir que os novos usuários compreendam as operações básicas antes de prosseguir para o uso prático do sistema.

Na tela inicial, caso o usuário opte por criar uma conta, ele será redirecionado para uma interface específica de cadastro. Nessa etapa, o usuário insere dados pessoais, como nome

completo e endereço de e-mai e cpfl, além de criar uma senha de acesso. Para garantir a validade do cadastro, o sistema envia um e-mail de confirmação contendo um link, que o usuário deve acessar para validar sua conta antes de continuar. Após a confirmação, a conta é ativada, e o usuário pode acessar as demais funcionalidades do aplicativo.

Após confirmar o email e ter a conta criada, o usuário deve se autenticar utilizando a funcionalidade da assinatura digital. É enviado um email para o usuário com o pdf da declaração que deve ser assinado no GovBR. Após o usuário providenciar o documento assinado, o aplicativo faz a verificação de identidade do usuário, comparando nome completo e cpf do cadastro com o do certificado da assinatura. Após isso, a conta é considerada autenticada. Com a conta autenticada, o sistema permite ao usuário cadastrar dados adicionais, como números de telefone, endereços de e-mail ou perfis em plataformas digitais.

O sistema oferece telas de busca e verificação de perfis. O usuário, mesmo sem estar autenticado, pode pesquisar por um usuário pelo nome completo e o selecionar, após isso é possível verificar dados como um número de telefone ou outro dado para verificar se ele corresponde a um perfil registrado. Os resultados dessa busca são apresentados em telas específicas, que indicam se os dados conferem, não conferem ou não estão cadastrados. Este fluxo garante que o aplicativo seja utilizado de maneira eficaz para validar informações com base nos registros disponíveis.

Figura 15 - Ícone do app

Figura 16 -Telas de tutorial

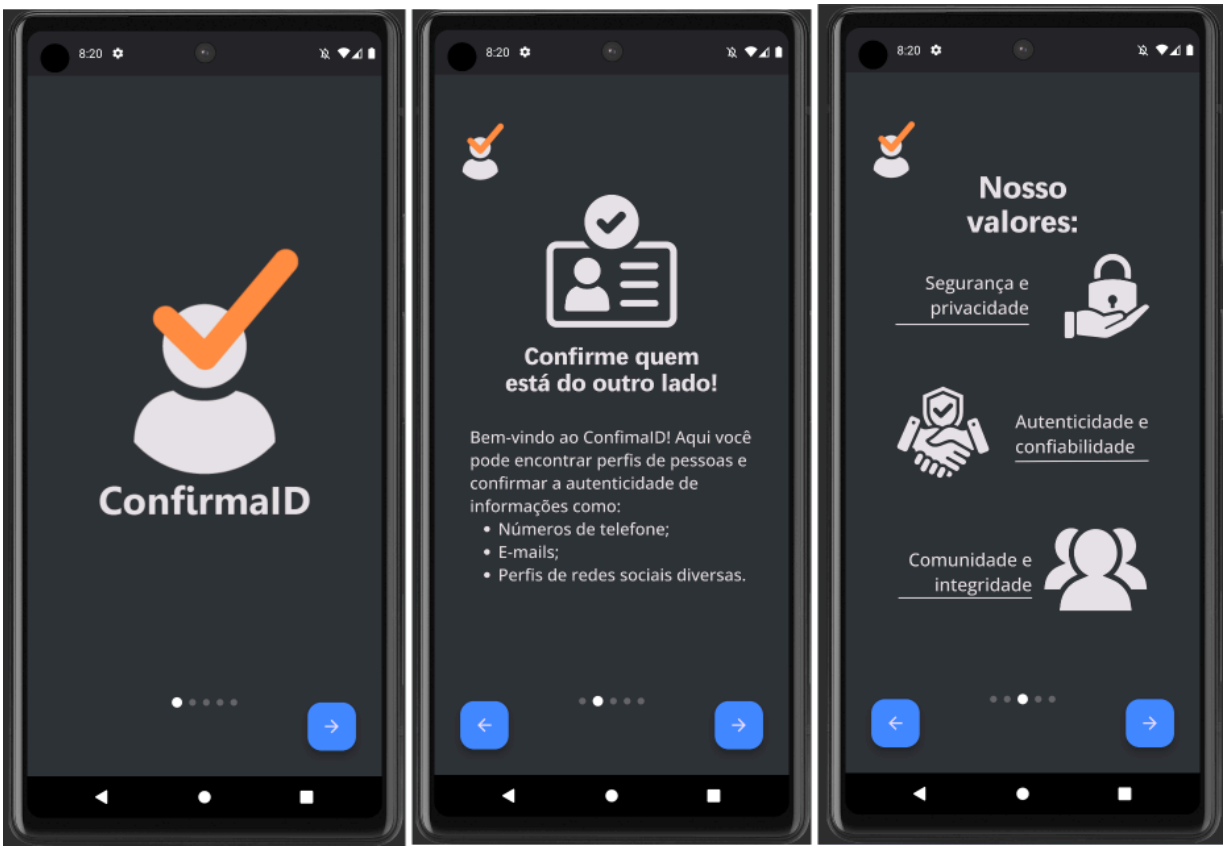


Figura 17 -Telas de tutorial

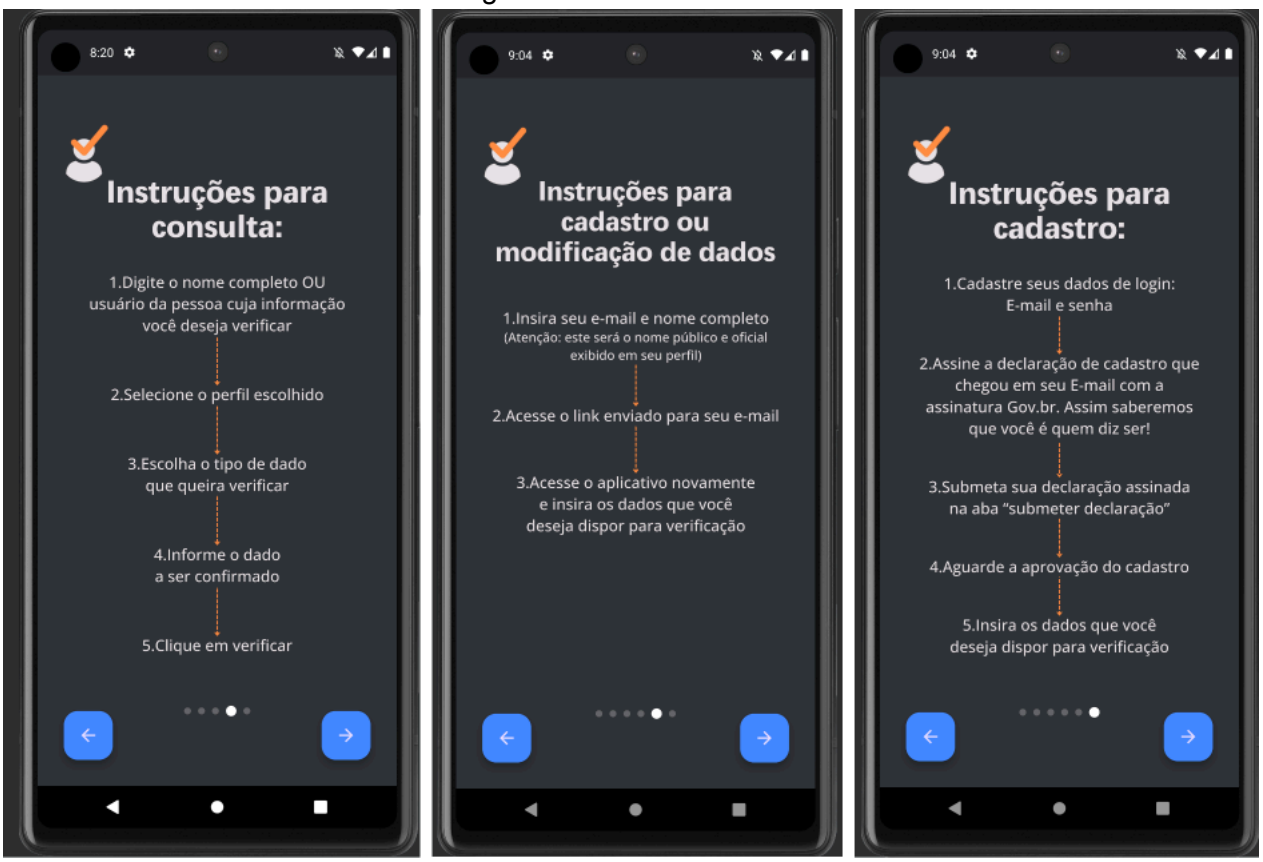


Figura 18 - Tela inicial e tela de criação de conta

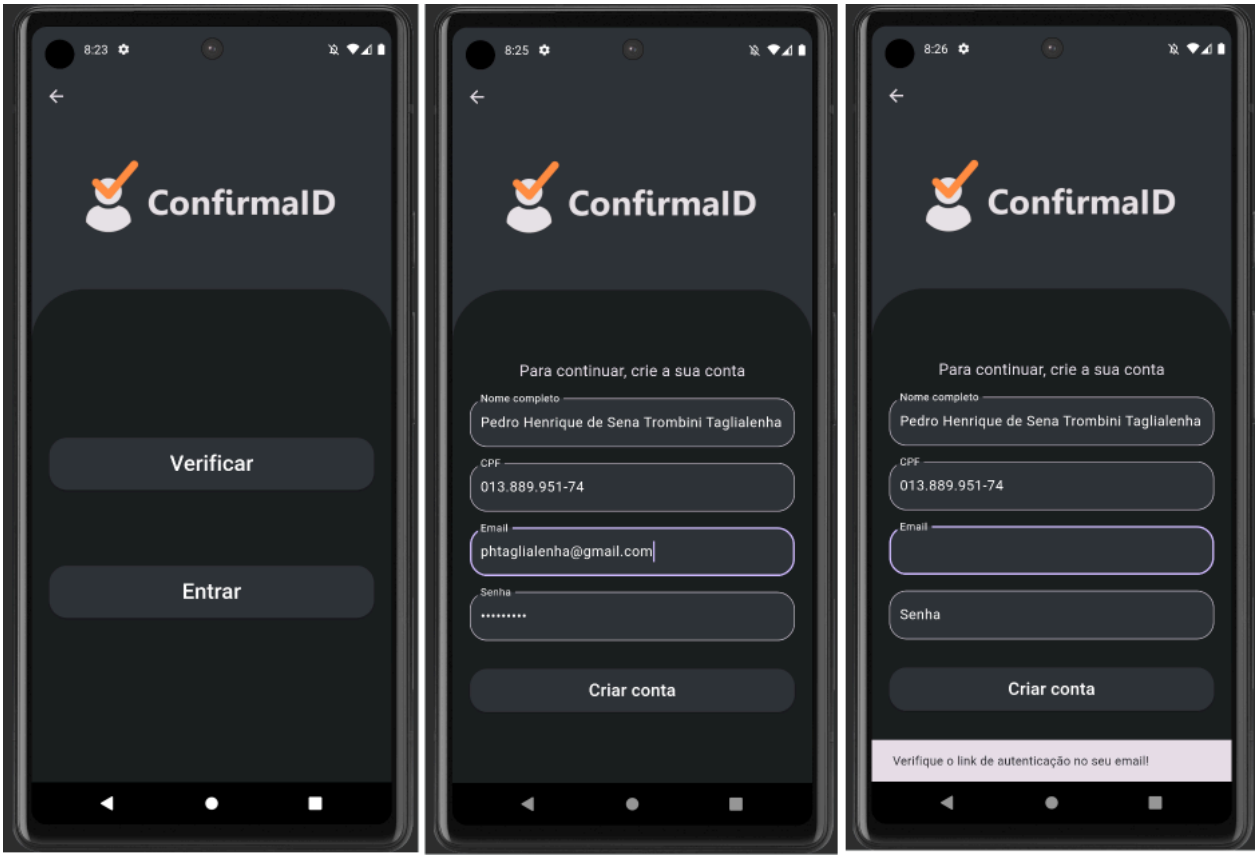


Figura 20 - Confirmação por email, telas de login e tela de autenticação por assinatura digital

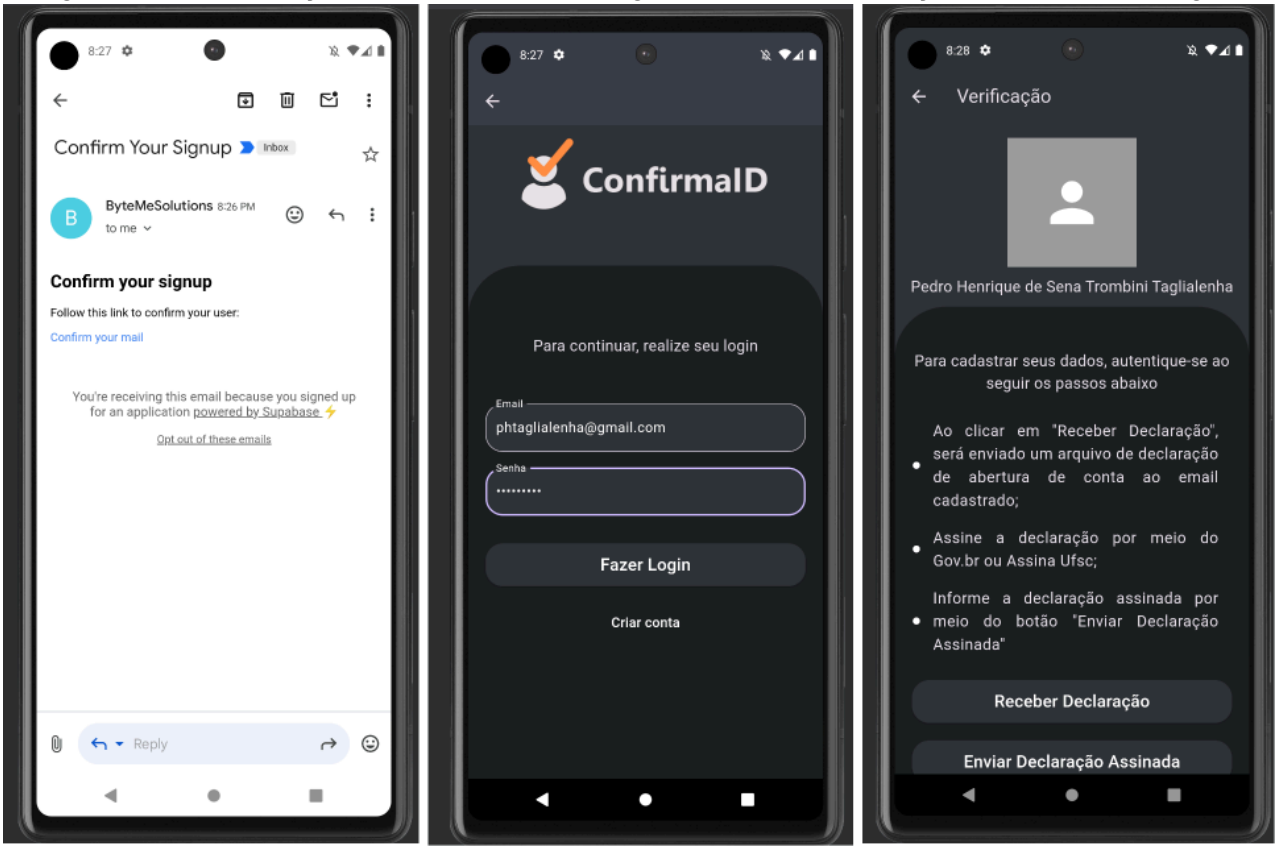


Figura 21 - Email para baixar o pdf da declaração e assinatura pelo GovBR da declaração

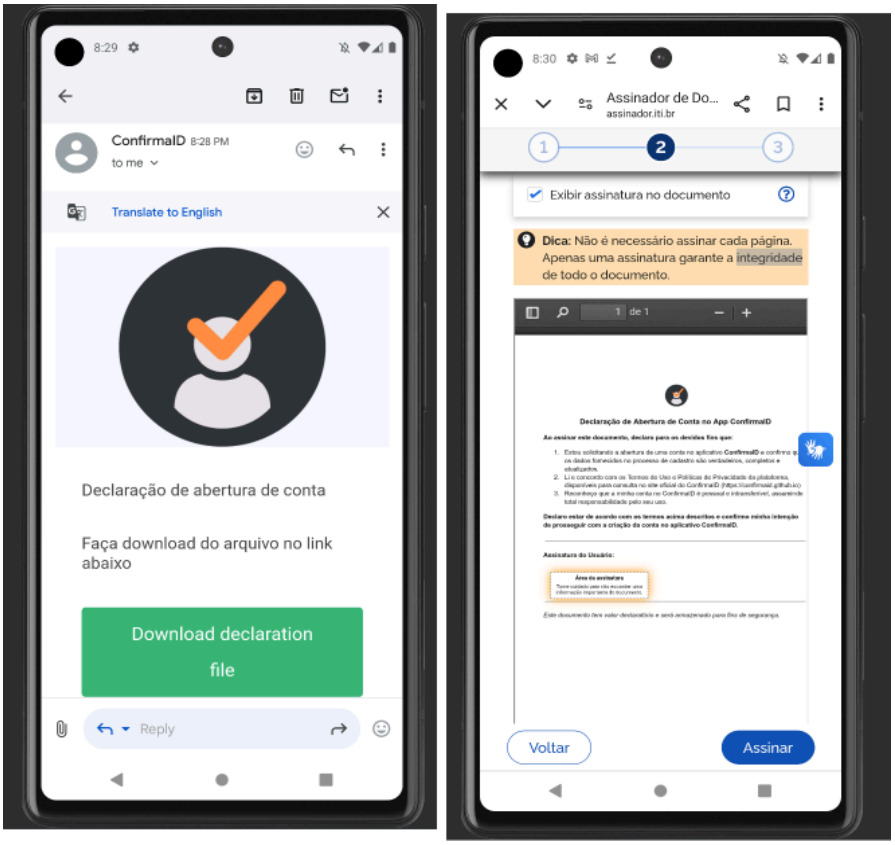


Figura 22 - Telas de cadastro de dados a partir da tela inicial

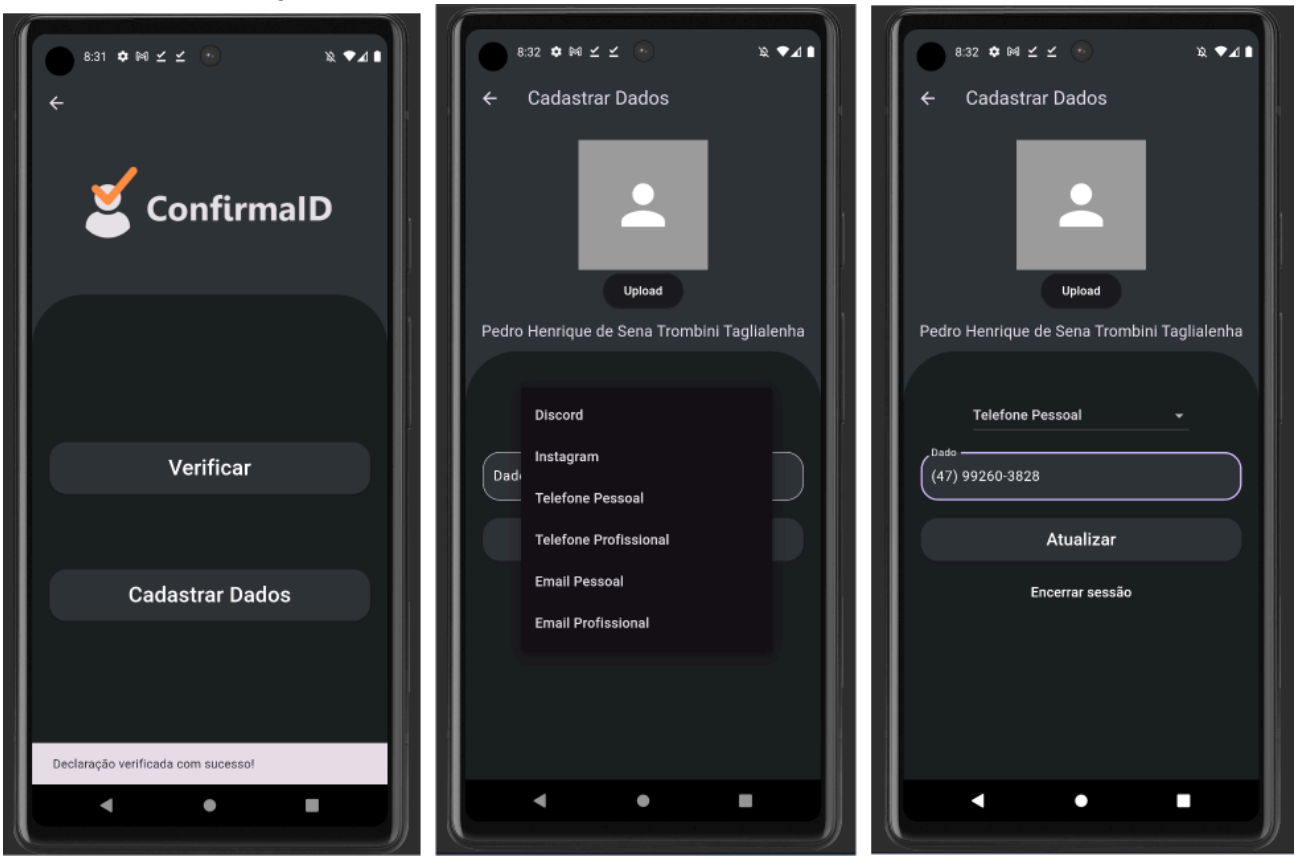


Figura 23 - Telas de busca de perfil e verificação com o número de telefone incorreto

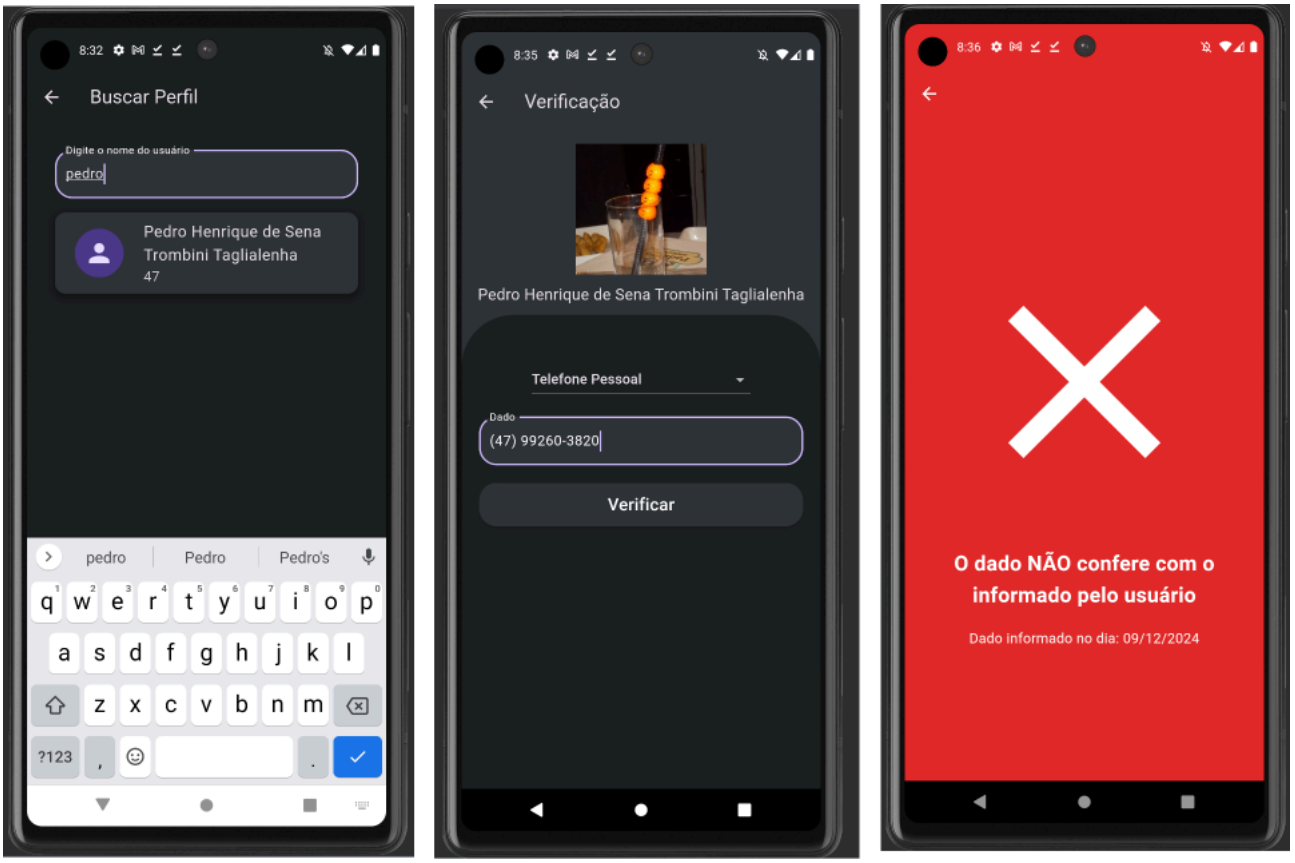
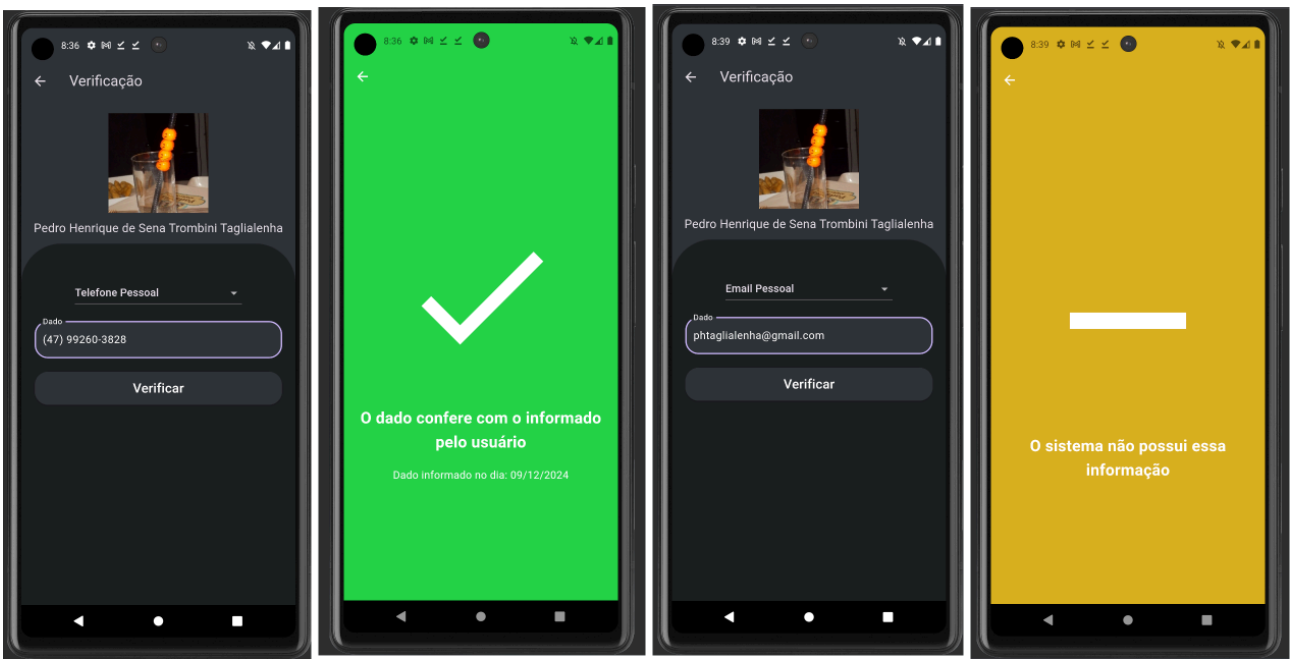


Figura 24 - Telas de verificação com o número correto e com um dado não cadastrado



7.4 USO DE SISTEMAS EXTERNOS:

a) APIs externas:

As APIs externas utilizadas na implementação as funcionalidades do aplicativo foram:

- Verificador de Conformidade ICP-Brasil[10]: Utilizado para automatizar a verificação das assinaturas digitais dos documentos de declaração de abertura de conta submetidos pelos usuários.
- NotificationAPI[11]: Utilizado para automatizar o envio do documento de declaração de abertura de conta para o e-mail do usuário.

b) Sistema de backend:

- Supabase[12]: Plataforma open source de backend como serviço (BaaS - Backend as a Service) com banco de dados, baseada no PostgreSQL. Fornece também integração com o SMTP do Google para envio de mensagem de e-mail para autenticação do e-mail de login.

c) Site hospedado no Github[13]:

- Para garantir que os usuários tenham acesso fácil aos nossos termos de uso e a informações básicas sobre o sistema, desenvolvemos um site hospedado no GitHub, estruturado com HTML.

[10] Documentação da API do Verificador de Conformidade ICP-Brasil
[11] NotificationAPI
[12] Supabase
[13] https://confirmaid.github.io/

8 TRABALHO EM GRUPO E DIFICULDADES ENCONTRADAS

O trabalho foi desenvolvido em grupo, com a participação de três integrantes: Pedro, Vitor e Rita. Pedro e Vitor ficaram responsáveis pelo desenvolvimento técnico do aplicativo, utilizando o framework Flutter, APIs externas e banco de dados para implementar as funcionalidades previstas. Rita contribuiu com o design do projeto e produto, a elaboração da fundamentação teórica e no uso de sistemas e APIs externos. A divisão de tarefas foi bem equilibrada, não deixando nenhum indivíduo sobrecarregado ou sem atividades para fazer. Essa abordagem proporcionou uma experiência enriquecedora e proveitosa para todos os envolvidos.

As principais dificuldades encontradas durante o projeto estiveram relacionadas à sua complexidade, que acabou sendo mais ambicioso que inicialmente previsto, e ao prazo limitado para entrega, agravado pelo fato de seu início coincidir com o período de provas em outras disciplinas. O desenvolvimento técnico apresentou desafios significativos, como a implementação de mecanismos para validar a identidade dos usuários durante o cadastro, uma etapa crítica para a segurança do aplicativo. A comunicação entre os integrantes, embora inicialmente desorganizada, foi aprimorada ao longo do trabalho, o que resultou em maior alinhamento e eficiência na execução das tarefas.

9 CONCLUSÃO

O aplicativo ConfirmaID ilustra como princípios fundamentais de segurança da informação, como assinaturas digitais e funções de hash, podem ser aplicados para mitigar riscos associados à fraude de identidade em interações digitais. A abordagem adotada para validar identidades por meio de certificados digitais vinculados ao Gov.br demonstrou ser uma solução prática e confiável para os desafios de autenticação. O uso de uma arquitetura modular facilitou a manutenção e a evolução do sistema, enquanto o armazenamento seguro de dados sensíveis através de hashes reforçou a privacidade e a conformidade com regulamentos como a LGPD.

Embora o sistema tenha se limitado ao escopo de um modelo mínimo viável, ele oferece uma base sólida para expansão e aprimoramento. A integração de técnicas adicionais, como proteção contra ataques de repetição, e a exploração de novos modelos de autenticação digital podem ampliar a aplicabilidade da solução.

10 REFERÊNCIAS

[1] TCC Victor Hardt *"Análise de Usabilidade do Aplicativo Móvel Medida Inteligente"*

[2] Cryptography and Network Security Principles and Practice, Stallings, 2004

[3] ©2024 Washington University in St. Louis, office of Information Security, Integrity. Acesso em: Integrity | Office of Information Security | Washington University in St. Louis

[4] Handbook of Applied Cryptography by A. Menezes, P. van Oorschot and S. Vanstone.

[5] J. L. Camp, "Digital identity," in IEEE Technology and Society Magazine, vol. 23, no. 3, pp. 34-41, Fall 2004, doi: 10.1109/MTAS.2004.1337889.

[6] Jøsang A. at al, “User Centric Identity Management”, in AusCERT Conference, 2005

[7] “Digital Identity”, Site: Definition of Digital Identity | BeyondTrust. Acessado em: 31/11/2024

[8] “Understanding digital identity theft and fraud”. Site: Understanding digital identity theft and fraud Acessado em: 31/11/2024

[9] “O que é phishing?” Site: O que é phishing? | IBM Acessado em: 31/11/2024

[10] “O que é catfishing e o que você pode fazer se for vítima?“ Site: O que é catfishing e o que você pode fazer se for vítima? | CNN Brasil Acessado em: 31/11/2024

[11] “O que é um ataque de roubo de conta?“ Site: O que é um ataque de roubo de conta? Acessado em: 31/11/2024

[12] Slide disponibilizado no moodle “Gestão de Identidades” Prof. Jean Everson Martina. Disponível em: https://presencial.moodle.ufsc.br/pluginfile.php/625550/mod_resource/content/2/5%20-%20INE5429%20-%20Gesta%CC%83o%20de%20Identidades.pdf

[13] “Saiba mais sobre a Assinatura Eletrônica“ Site governamental: Saiba mais sobre a assinatura eletrônica — Governo Digital Acessado em: 30/11/2024

[14] “Assinatura eletrônica avançada“ Site governamental: Assinatura eletrônica avançada Acessado em: 30/11/2024

[15] Anagu, E. J et al, “Verified Views: How Blockchain-enabled Digital Identity Verification Can Combat Fake Accounts and Disinformation on Social Media”, in International Journal of Emerging Multidisciplinaries, 2024