UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Ricardo Pereira

Desenvolvimento do *Multifactor*: Aplicação mobile que implemente serviços de verificação de duas etapas para autenticação multifator

Uberlândia, Brasil

2022

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Ricardo Pereira

Desenvolvimento do *Multifactor*: Aplicação mobile que implemente serviços de verificação de duas etapas para autenticação multifator

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Orientador: Pedro Frosi Rosa

Universidade Federal de Uberlândia – UFU

Faculdade de Ciência da Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2022

Ricardo Pereira

Desenvolvimento do *Multifactor*: Aplicação mobile que implemente serviços de verificação de duas etapas para autenticação multifator

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, Minas Gerais, como requisito exigido parcial à obtenção do grau de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 25 de julho de 2022:

Dr. Pedro Frosi Rosa
Orientador

Dr. Rodrigo Sanches Miani
Professor

Dr. Flávio de Oliveira Silva
Professor

Uberlândia, Brasil
2022

Resumo

Este trabalho tem por objetivo a construção de uma aplicação mobile para autenticação multifator com o intuito de permitir que usuários possam adicionar um camada extra de proteção ao ratificar seu acesso em qualquer serviço que ofereça suporte a esta tecnologia, de forma prática e gratuita. O uso de autenticação com 2 ou mais fatores se faz cada vez mais necessário uma vez que *cyber* ataques são um problema de nível global no setor de segurança de informação. Atacantes desfrutam de técnicas cada vez mais refinadas e geram a governos, instituições financeiras, demais tipos organizações e usuários da internet prejuízos que chegam à casa de bilhões de dólares.Nesta perspectiva o objetivo principal deste trabalho é o de proteger informações sensíveis de usuários de possíveis atacantes, em especial, vítimas de ataques de *phishing*, proporcionando aos mesmos o benefício de ter seus dados seguros mesmo que suas credencias de alguma forma sejam roubadas.

Palavras-chave: Autenticação multifator, *MFA*, *Phishing*, *OTP*, *HMAC*, Segurança da informação.

Lista de ilustrações

Lista de abreviaturas e siglas

OTP	*One time password*
HMAC	*keyed-hash message authentication code*
MFA	*Multi-factor authentication*
APP	*Application*
PIN	*Personal Identification Number*
TOTP	*Time-based One-Time Password*
MVC	*Model - View - Controller*
APWG	*Anti-Phishing Working Group*
SIM	*subscriber identity module*
HTTPS	*Hyper Text Transfer Protocol Secure*
IDE	*Integrated Development Environment*
2FA	*Two Factor Authentication*
CLI	*command-line interface*
WEB	*World Wide Web*

Sumário

1 INTRODUÇÃO

A autenticação de dois ou mais fatores é um método extremamente eficaz em complementar a segurança de aplicações uma vez que este método usufrui de técnicas de *cyber* segurança para impedir que um potencial atacante consiga acessos a informações sensíveis de um usuário ainda que o invasor possua as credenciais de acesso da vítima. Melhorar a segurança na autenticação de usuários tendo em vista inibir acessos indevidos a sistemas de informação governamentais, bancários ou de organizações prestadoras de serviço é fundamental o usuário final destas entidades.

Senhas de tipo único ou também conhecidas como senhas descartáveis são palavras chave ou *PIN* gerados de forma automática por um chaveiro *OTP* ou alguma aplicação. Estes são um nível adicional de segurança para *MFA* pois independente da forma com que foram gerados, não podem ser reutilizados e são exigidos toda vez que um usuário se conecta a algum serviço, impedindo ataques de *replay* e seu uso basicamente consiste em garantir que um usuário a cada acesso em alguma plataforma informe um *PIN* diferente.

Códigos de autenticação de mensagens com chave *hash* (*HMAC*) acrescentam mais uma camada de segurança a autenticação múltiplos fatores. Estes códigos são nada mais do que o resultado de uma função que utiliza como parâmetros um segredo (utilizado para cálculo de uma chave interna e externa) e a mensagem na qual deseja-se garantir sua integridade e autenticidade. Esta função não irá criptografar o recado, mas sim calcular uma *hash* do mesmo com base nas chaves criadas, dessa forma envia-se a mensagem juntamente com o resultado da função, assegurando que o receptor da mensagem calcule novamente o *hash* do recado. Quando o resultado do *hash* calculado pelo lado do receptor é isomorfo ao contido na mensagem significa que a informação é integra e autêntica. *HMAC* é facilmente adaptável para uma aplicação assegure que um determinado segredo ou *PIN* *OTP* seja de uma conte confiável e não fraudulenta.

1.1 Objetivos

Projetar e desenvolver uma aplicação mobile que implemente um serviço de autenticação de usuários em duas etapas utilizando senhas descartáveis (*OTP*) geradas em um determinado intervalo de tempo e desfrutando também de senha única baseado em código de autenticação de mensagens com chave *hash* (*HMAC*) para autenticar usuários em aplicações de software.

1.1.1 Objetivos específicos

1. Modelagem de aplicação mobile para verificação em duas etapas;
2. Desenvolvimento de aplicação mobile para oferecer serviço de verificação em duas etapas;
3. Disponibilização do *APP* para os sistemas operacionais *Android* e *IOS*.

1.2 Organização do trabalho

Desenvolver uma aplicação mobile que implemente um serviço de autenticação de usuários em duas etapas utilizando senhas descartáveis (OTP) geradas em um determinado intervalo de tempo e desfrutando também de senha única baseado em código de autenticação de mensagens com chave *hash* ($HMAC$) para autenticar usuários em aplicações de software.

2 FUNDAMENTAÇÃO TEÓRICA

Atualmente para que um processo de autenticação seja considerado seguro deve ser levado em conta que mesmo um atacante descubra as credenciais de acesso de algum usuário por meio de roubo de credenciais ou espionagem da comunicação do usuário com a aplicação, o acesso as informações esteja restrito. A autenticação multifator consegue garantir este processo, contudo é preciso assegurar que o método criptográfico seja confiável, seguro, unidirecional e permita ser implementado do lado do usuário em um computador, *smartphone*, microcomputador ou outro dispositivo (ALOUL WASSIM EL-HAJJ, 2009).

Considerando que um autenticador multifator deve garantir que o processo não seja fraudulento, algumas medidas devem ser adotadas como o fato de o OTP ser gerado somente no *smartphone* do usuário e utilizando o cartão SIM, data ou tempo como base para gerar uma senha de uso único único que não deve ter custo para sua geração e jamais ser armazenada no dispositivo do usuário.A aplicação deve ser executada facilmente em quaisquer sejam as plataformas de sua implementação, não precisar de nenhuma conexão com algum servidor para funcionamento e possuir uma interface amigável com o usuário final (LAMPORT, 1981).

Nos últimos tempos ataques de *phishing* destacam-se por ser um dos ataques mais enfrentados por usuários da Internet, governos e corporações. A defesa contra ataques desta natureza ainda é considerado por especialistas em segurança um desafio, das abordagens existentes algumas são baseadas algoritmos de aprendizado de máquina, contudo apesar de oferecerem um excelente resultado, não são perfeitas e a detecção de ataques *phishing* por tais técnicas ainda fornece altas taxas de falsos positivos (BASIT A., 2021).

Segundo o relatório de investigação de violação de dados divulgado pela *Verizon*, no ano de 2020 o número ataques baseados em engenharia social como *pretexting* e *phishing* dominaram o cenário de crimes cibernéticos. A motivação por trás deste tipo de ataque é quase sempre financeira, porém o estudo afirma que os criminosos não tem se dado o trabalho de inventar um novo cenário, tendendo a ser eficientes com esforços mais básicos. Autenticação com 2 ou mais fatores possuí um alto gral de eficácia na proteção das informações do usuário final, permitido que mesmo com o vazamento de seus dados, o tempo de resposta a situação seja o suficiente para evitar efeitos colaterais indesejados (VERIZON, 2021).

Conforme os resultados apresentados pelo relatório da $APWG$ (*Anti-Phishing Working Group*), no ano de 2020,a *Axur*, empresa membro do $APWG$ sediada no Brasil aponta que o setor bancário e financeiro ainda é o principal alvo dos ataques de *phishing* no país. Na tentativa de evitar a detecção pelos defensores, boa parte dos criminosos passaram a

registrar domínios que não continham uma palavra-chave atraente projetada para enganar os consumidores além de uma fatia considerável dos sites utilizados para fraudes eram protegidos pelo protocolo de criptografia HTTPS na tentativa de se camuflar (APWG, 2020).

Existem diversas formas de se desenvolver um gerenciador de senhas de tipo único, e cada vez mais aplicações tecnologias que compõem aplicações híbridas como o *IONIC* se tornam vantajosas por apresentarem uma excelente robustez permitindo a construção dos mais diversos nichos de aplicações aplicações.(RABOY, 2016). Algorítimos para geração de senhas de tipo único atualmente podem ser encontrados em bibliotecas de diversas linguagens, dente elas o *javaScript*, que será a base da construção do *Multifactor*. A construção de um *OTP* utilizando a linguagem escolhida é relativamente simples e extremamente eficaz por permitir que código seja executado do lado do usuário final em diversas aplicações sem muita dificuldade (RABOY, 2014).

3 Metodologia

Este capítulo apresenta a metodologia utilizada para implementação desta aplicação em conjunto com todas as ferramentas necessárias para seu desenvolvimento e compilação, bem como as tecnologias embarcadas no mesmo.

3.1 Ferramentas de desenvolvimento

3.1.1 *Android Studio*

O *Android Studio* é um de Ambiente de Desenvolvimento Integrado ou *IDE*, sigla em inglês para *Integrated Development Environment*. Seu principal recurso é seu ambiente unificado que permite o desenvolvimento de aplicações aos inúmeros dispositivos *Android* existente e suas diversas versões do sistema operacional.

3.1.2 *Xcode*

Xcode é um ambiente de desenvolvimento integrado e seu principal recurso é permitir um ambiente unificado com todas as ferramentas necessárias no desenvolvimento de qualquer plataforma da *Apple*.

3.2 Tecnologias

Será apresentado nesta seção todas as tecnologias, bibliotecas e *frameworks* utilizadas para construção desta aplicação, assim como sua devida importância no mesmo. O desenvolvimento deste projeto é realizado utilizando a *framework IONIC* e se divide primordialmente em duas fazes: A primeira que denominamos de *back-end* na qual foi utilizado *TypeScript*, *SQLite*. A parte de *front-end* engloba tecnologias como *HTML*, *CSS*, *JavaScript* e *AngularJS*.

3.2.1 *Ionic*

Ionic é uma *framework* que permite o desenvolvimento de aplicações mobile utilizando *HTML5*. Sua utilização é focada para a construção de aplicativos móveis híbridos, que em sua essência são pequenos sites executados em um *shell* de algum navegador num aplicativo que tem acesso à camada nativa da plataforma do qual foi compilado. *APPs* com esta característica têm muitos benefícios em relação aos aplicativos nativos, em es-

pecial com relação a termos de suporte à plataforma, velocidade de desenvolvimento e acesso a código de terceiros.

3.2.2 *Capacitor*

Capacitor é um *framework* de código aberto feito para para criar aplicativos nativos da *Web*. Possui um sistemas de *plugins* que possibilita a comunicação entre código *JavaScript* da aplicação com funcionalidades nativas da plataforma do qual foi compilado. Tal competência possibilita um processo de desenvolvimento mais simples, que também oferece suporte a tecnologias atualizadas.

3.2.3 *TypeScript*

TypeScript é una linguagem de programação fortemente tipada que se baseia em *JavaScript*. Foi criada no ano de 2012 pela *Microsoft*, seu intuito é permitir que o *JavaScript* tenha melhor suporte ao paradigma de Programação Orientada a Objetos, agregando maior robustez na linguagem.

3.2.4 *HTML*

HTML ou *HyperText Markup Language*, que traduzido para o português significa "Linguagem de Marcação de Hipertexto"é uma linguagem de marcação bastante utilizado para construção de páginas na *Web*.

3.2.5 *CSS*

CSS (*Cascading Style Sheets*) é um mecanismo para adicionar estilos como cores, fontes, espaçamento, entre outros a uma pagina web. Pode ser aplicado diretamente nas *tags HTML* ou ficar contido dentro de um arquivo de estilo *CSS*.

3.2.6 *JavaScript*

JavaScript é uma linguagem leve, interpretada e baseada em objetos, muito conhecida como uma linguagem de script para páginas *Web*. Suporta estilos de programação orientados a eventos, funcionais e de fácil aprendizagem.

3.2.7 *Angular*

Angular foi criado pelo *Google* e pode ser considerado como uma *framework* de código aberto, escrito em *javaScript* para o desenvolvimento de aplicações *web*.

4 Arquitetura

Neste capítulo é apresentado a arquitetura utilizada para implementação da aplicação, bem como seu funcionamento no mesmo. Conforme pode ser visto na figura 1 a aplicação *Multifactor* é um *APP* híbrido cujo *Front-End* é composto essencialmente por tecnologias *WEB* como *HTML*, *CSS* e *JavaScript*, que interage com seu *Back-End* através do *TypeScript* que compõem em essencial a regra de negócio da aplicação utilizando componentes, templetes e serviços que são um conjunto de ferramentas providos pelo *AngularJS* para auxílio no desenvolvimento de uma aplicação.

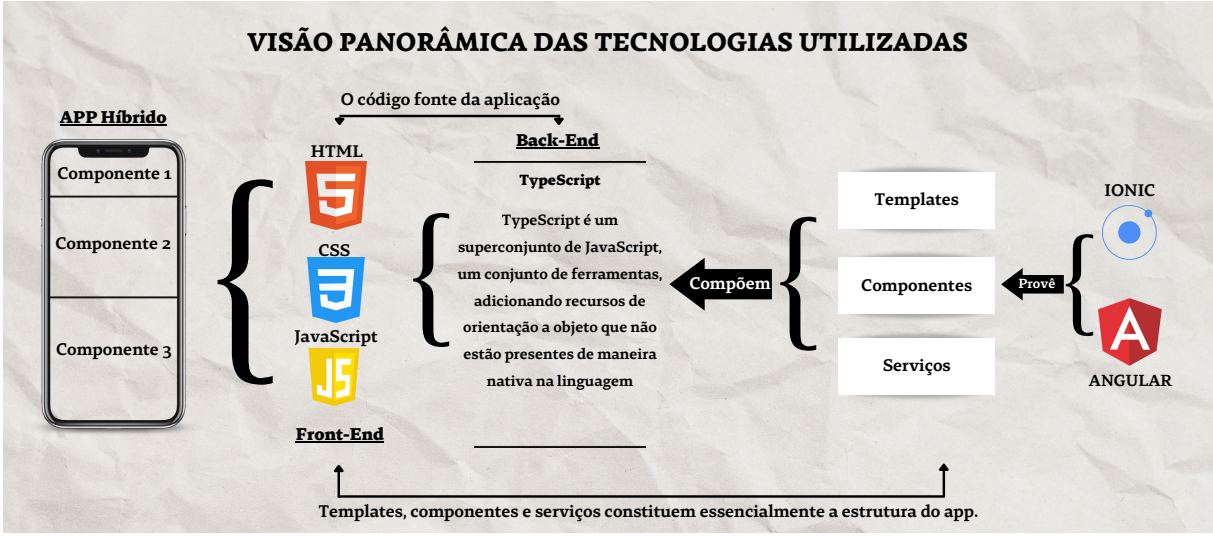

Figura 1 – Visão panorâmica das tecnologias utilizadas

4.1 Padrão de arquitetura

Neste capítulo será apresentado um pouco sobre a arquitetura escolhida para ser utilizada na construção do *Multifactor*, explorando não somente a parte conceitual do padrão de projeto *MVC* mas também explorando o seu comportamento na aplicação de forma prática.

Através de um diagrama de sequências básico será demonstrado como o usuário final pode utilizar a aplicação para inserir novas credenciais de serviços que o mesmo utiliza, assim como sua atuação no cotiano do mesmo para realizar autenticação de 2 ou mais fatores em aplicações *WEB*.

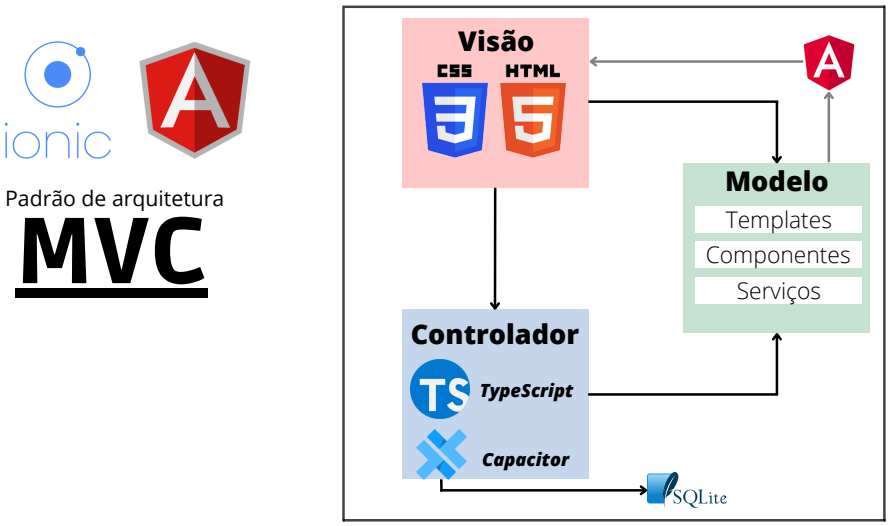

Figura 2 – Padrão de arquitetura *MVC*

4.1.1 Padrão de arquitetura *MVC*

Em aplicações que utilizam *AngularJS* como um dos pilares de execução da aplicação possuem um ciclo de vida simples e que permite adotar diversos padrões de projeto. O paradigma escolhido foi o *MVC* (*Model-View-Controller*), cujo foco consiste na reutilização de código, o dividindo em camadas interconectadas denominadas de modelo, visão e controle.

O modelo pode ser classificado como o elo entre a camada de visão e a camada de controle. Seu papel na aplicação consiste no gerenciamento dos dados através de das regras de negócio, permitido que o mesmo insira, atualize ou delete dados através de comandos realizados da camada de controle.

A visão pode ser considerada como a camada cujo os dados solicitados do modelo são exibidos ao usuário final, permitido que o mesmo interaja com com estas informações através de listas, botões entre outros. Primordialmente uma visão possui a responsabilidade de garantir que sua instância reflita de forma consistente as informações contidas na camada de modelo e quando há alguma alteração no mesmo a camada de visão possui a possibilidade de se atualizar através da camada de controle.

A camada de controle que faz a interação de entrada e saída de dados da aplicação através de comandos nas camadas de visão e modelo, para que sejam alteradas de forma apropriada para o usuário final. O principal foco da camada de controle é a manipulação de dados com base nas ações do usuário. Em resumo é possível alegar que o Controle envia ações para o Modelo e para a visão onde serão realizadas as operações necessárias.

4.1.2 Atuação do *MVC* no *Multifactor*

No Multifactor a camada de visão atua de forma simples, permitindo que o usuário final tenha visão clara dos segredos descartáveis gerados através serviços cadastrados na aplicação e seu tempo de duração, permitindo que seja possível inserir ou remover estas credenciais registradas através de uma interface gráfica.

A camada de controle atua principalmente interagindo com a camada de visão, cuja repesabilidade é assegurar que qualquer modificação sofrida no modelo seja representada de forma clara e que faça sentido para o usuário final, uma vez que os segredos descartáveis possuem um tempo de via útil curto e mudam periodicamente. Toda vez em adicionada e excluída, cabe a controladora fazer o manejo adequado das informações.

O modelo atua em geral na regra de negócio da aplicação, contendo toda a estrutura que permita inserir, remover, atualizar ou listar registros do banco de dados. Pode conter também métodos fundamentais na composição da regra de negócio, no caso do *Multifactor* temos a possibilidade de gerar *OTP's* das credenciais registradas no banco de dados através dos serviços, componentes ou templetes disponibilizados pelo *AngularJS*.

4.1.3 Configurando autenticação de 2 fatores em alguma aplicação utilizando o *Multifactor*

Utilizar o *Multifactor* para realizar autenticação de 2 ou mais fatores em alguma aplicação não é algo complexo, em geral usuário não precisa realizar muitos procedimentos. Como passo inicial o mesmo precisa realizar login na mesmo com o método de autenticação convencional, utilizando sua senha e a plataforma iniciar a configuração de autenticação em 2 etapas.

Iniciada a etapa de configuração de *2FA*, apesar de existirem exceções, boa parte das aplicações utilizarão em esquema semelhante ao apresentado na imagem abaixo, gerando uma *token* que comumente nos dias de hoje pode variar até o máximo de 320 bits. Esta credencial é enviada pelo usuário que registra a mesma no *Multifactor* que a partir deste momento passará a gerar senhas descartáveis do serviço.

Posterior a configuração da credencial no *Multifactor*, é comum as aplicações solicitem ao usuário a inserção algum *OTP* gerado para validar o mesmo concluir o processo de configuração. Esta etapa é importante pois garante primeiramente que o usuário foi bem sucedido em concluir a configuração de registro da credencial em alguma aplicação. É preciso assegurar também que a senha descartável entregue pelo usuário seja válida, tenha o comprimento esperado pela aplicação, assegurando que o *OTP* foi gerado com algum algorítimo compatível.

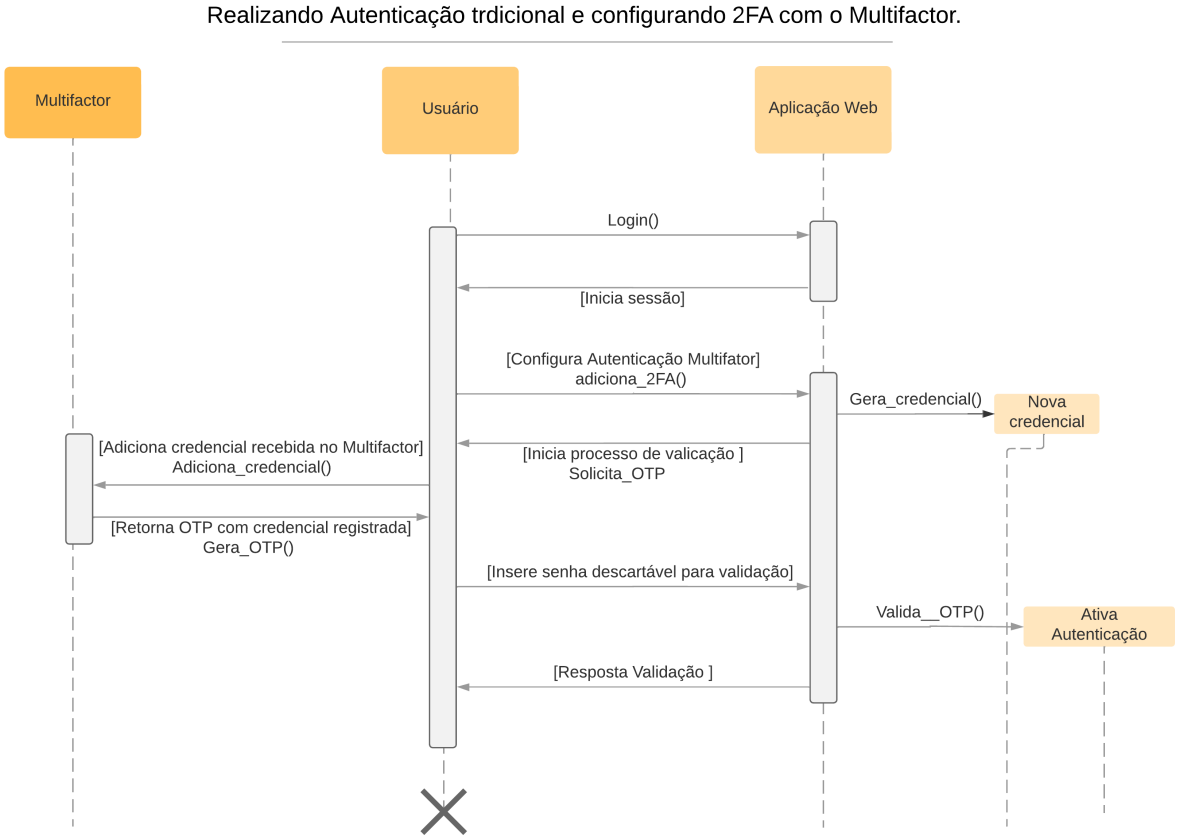

Figura 3 – Diagrama de sequência de um usuário configurando 2FA em alguma aplicação utilizando o *Multifactor*

4.1.4 Autenticação de 2 fatores em alguma aplicação utilizando o *Multifactor*

Para que um usuário possa realizar login em alguma plataforma e utilize o *Multifactor* como forma de autenticação de 2 fatores, partimos do pressuposto de que a etapa de configuração vista na imagem anterior tenha sido cumprida. O processo neste tipo de situação é relativamente mais simples, contudo há pontos a serem ressaltados em cada etapa.

Após realizar a autenticação tradicional é subentendido que um segundo fator de autenticação será solicitado pela aplicação cujo usuário está realizando login, porém é importante ressaltar que o mesmo precisa apenas possuir o aparelho disponível e em mãos e entregar na aplicação a senha descartável para ser validada e permitir seu acesso no mesmo.

Este modelo de autenticação não tende a falhar, mesmo que o usuário esteja sem conexão com a internet ou em localidades com fuso horário do que registrou o *OTP* não acarretarão em resultados diferentes para o cálculo das senhas descartáveis. O único elo que o *Multifactor* não pode cobrir são sinistros externos cujo *APP* não pode impossibilitar

como a perda do aparelho celular, indisponibilidade de algum serviço e etc. A desativação ou troca de credenciais para *2FA* varia de um serviço para outro, logo, ainda que a aplicação consiga assegurar uma camada extra de segurança, neste processo o de certa forma usuário ainda é incumbido de certas responsabilidades.

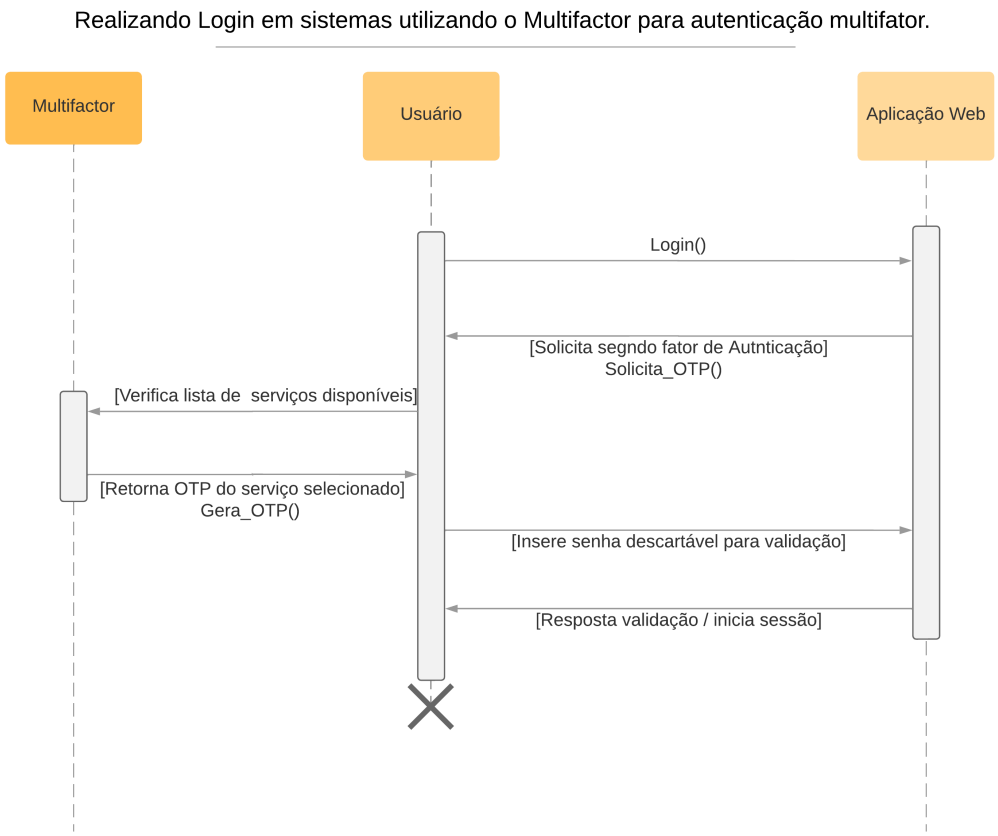

Figura 4 – Diagrama de sequência de um usuário realizando *login* em alguma aplicação utilizando o *Multifactor*

5 Desenvolvimento

Neste capitulo são abordados todos requisitos necessários para o construção da aplicação, desde as tenologias necessárias no ambiente de desenvolvimento em conjunto com suas respectivas versões. Também é apresentado como realizar a criação de um projeto *IONIC*, a instalação das dependências necessárias para o projeto, modelagem de seu armazenamento e os resultados obtidos no desenvolvimento do *APP*.

5.1 Requisitos do projeto

- *Node.js* versão 14.0 ou superior
- *Ionic* 6 *CLI*
- *Android Studio* ou *Xcode* instalado

5.2 Criando um projeto *IONIC*

Abra a linha de comando do seu Sistema operacional e Escolha o diretório do projeto. Após esta etapa digite o seguinte comando instalar o *Ionic framework*. Esta biblioteca é utilizada para executar binários nativos em dispositivos móveis ou simuladores/emuladores. O *cordova-res*, é usado para gerar ícones de aplicativos nativos e telas iniciais:

npm install -g @ionic/cli native-run cordova-res

Após esta etapa, no mesmo diretório digite o seguinte comando para criar um novo projeto *Ionic Angular* em branco com o nome de sua preferência:

ionic start nomeDoProjeto blank –type=angular

5.3 Instalando dependências do projeto

5.3.1 *SQLite*

Esta dependência permite que a aplicação crie e acesse o bancos de dados *SQLite* no dispositivo em que estiver instalado. É fundamental para armazenarmos os segredos no próprio dispositivo de forma segura e persistente, uma vez que utilizar o *localStorage* ou *sessionStorage* do *In App Browser* disponibilizado pelo *Ionic framework* está fora de

questão para maior segurança e persistência das informações a serem armazenadas, uma vez que qualquer limpeza de cache poderia apagar as credenciais armazenadas.

npm install cordova-sqlite-storage

npm install @awesome-cordova-plugins/sqlite

5.3.2 *Clipboard*

Este *plugin* que permite o gerenciamento de área de transferência do dispositivo em que a aplicação for instalada. Será utilizado para inserir na areá de transferência algum *OTP* gerado que o usuário selecionar para se autenticar em alguma plataforma.

npm install @ionic-native/clipboard

5.3.3 *Toast*

Este componente permite mostrar uma *toast* de forma nativa (pequeno *pop-up* de texto) no dispositivo em que a aplicação estiver instalada. É útil para mostrar uma notificação nativa que seja menos invasiva e direta para o usuário final, por exemplo quando algum *OTP* for selecionado pelo usuário e seu valor inserido na areá de transferência do dispositivo, é possível informar que o mesmo foi devidamente copiado.

npm install –save @ionic-native/toast@4

5.3.4 Jssha

Uma biblhoteca escrita em *TypeScript/JavaScript* que implementa de forma completa todas as *Secure Hash Standard* (SHA) com *HMAC*.

npm install jssha@1.6.2 –save

5.4 Criando componentes

Para este projeto iremos à pasta raiz do projeto, se dirigir a **./src/app/** e no mesmo criar um novo diretório com o nome ***services***. Em um projeto Angular, um serviço pode ser entendido como uma classe que atua como uma unidade da regra de negócios, onde é possível centralizar e reutilizar o código de maneira organizada.

Em um projeto *Ionic* os serviços não possuem visualização, apenas código reutilizável, sendo este o principal motivo pelo qual precisamos do mesmo nesta aplicação, pois pode ser importado em páginas, outros componentes e até mesmo em outros serviços. A característica de reutilização de serviços e componentes permite maior agilidade do desenvolvimento da aplicação.

5.4.1 Serviço *database*

O serviço *database* contém toda a regra de negócio que necessária para utilizar o banco de dados da aplicação, desde a criação do mesmo, assim como métodos para leitura, inserção e deleção de dados.

ionic generate service database

5.4.2 Serviço *password*

O componente *password* contém toda a regra de negócio que utilizaremos para calcular e gerar de forma correta os segredos descartáveis.

ionic generate service database

5.4.3 Armazenamento

Como mencionado anteriormente utilizamos o *SQLite* como banco de dados para esta aplicação. Seu modelo relacional é extremamente simples, uma que o *Multifactor* necessita somente de uma tabela para operar.

Sua única tabela se chama *passwords* e contém 3 campos sendo o primeiro deles o campo id que é do tipo inteiro, utilizado para identificar cada segredo armazenado no banco. O campo *title* é do tipo *text* e utilizado para nomear o sigilo a ser registrado. Por fim o campo *secret* que também é do tipo *text* que serve para armazenar o segredo a ser utilizado para geração do *OTP*.

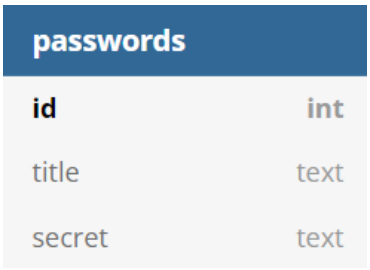

Figura 5 – Diagrama do modelo relacional do *Multifactor*

5.5 Resultados obtidos

Os resultados obtidos foram satisfatórios, foi desenvolvido uma tela inicial, onde o usuário tem a visualização das senhas descartáveis geradas periodicamente, assim como uma segunda tela somente para cadastros das mesmas. O algorítimo para produzir o *OTP* funcionou corretamente, sendo testado em diversos serviços.

5.5.1 Tela inicial do *Multifactor*

A tela inicial do *Multifactor* é simples, como apresentado a baixo, a mesma contém apenas uma listagem dos serviços cadastrados na aplicação em conjunto com seu *OTP*. Em cada item desta lista, ao dar um toque sobre o mesmo a senha de tipo único é copiada e enviada para a área de transferência do aparelho, esta operação também é acompanhada por uma *toast* informado o nome do serviço e senha copiados. Arrastando item da lista levemente para a esquerda é possível ter a cesso a opção de exclusão da credencial selecionada, aonde o serviço não teria mais senhas descartáveis gerados no *APP*. A parte inferior da tela contém uma barra aonde é informando o tempo restante cuja ainda são validas senhas instanciadas naquele momento e um botão que aciona a modal de registro de credenciais da aplicação.

O Interface do usuário escolhida para criação do *Multifactor* foi baseada em demais aplicações já existentes no mercado com o *Google Authenticator* e *Authy*. Este tipo de interface preza por ser minimalista, cuja intenção se tange em garantir que o usuário final consiga de forma prática realizar ações de cópia, deleção ou até mesmo acesso a registro de credenciais de novos serviços.

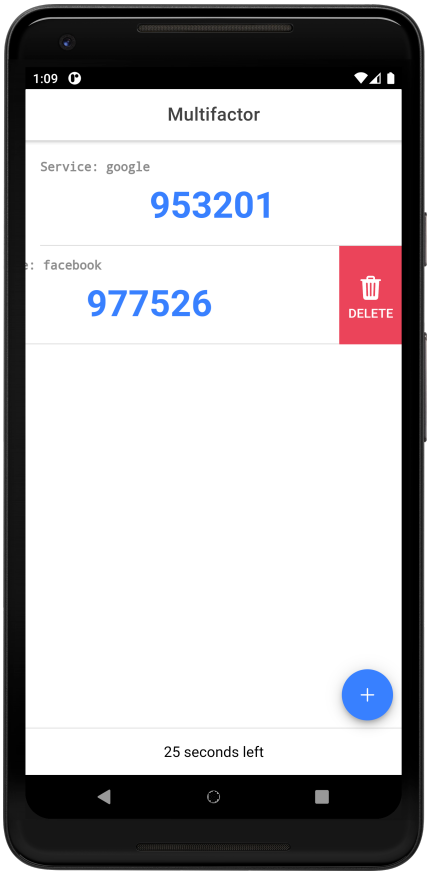

Figura 6 – Tela inicial do *Multifactor*

5.5.2 Tela de cadastro do *Multifactor*

Como apresentado na imagem abaixo a tela de cadastro do *Multifactor* segue os princípios adotados para a tela inicial, contendo apenas um fomulário simples cujo usuário fará o registro da nova Credencial e um botão na parte inferior para fechamento do moral.

O formulário realiza algumas validações, como o fato de não ser permitido o registro de uma credencial sem seu respectivo nome de serviço, assim como é realizado a tentativa de gerar um *OTP* com a credencial antes de seu registro, para garantir que credencias com tamanho ou formado não condizentes com o algorítimo utilizado precisem ser registrados no banco de dados para teste. Todos os avisos pertencentes a erros ou validações são exibidos para o usuário final através de uma caixa de alerta na própria tela, e caso tudo ocorra como o planejado a modal é fechada, tela inicial recarregada com o novo serviço já ativo na aplicação.

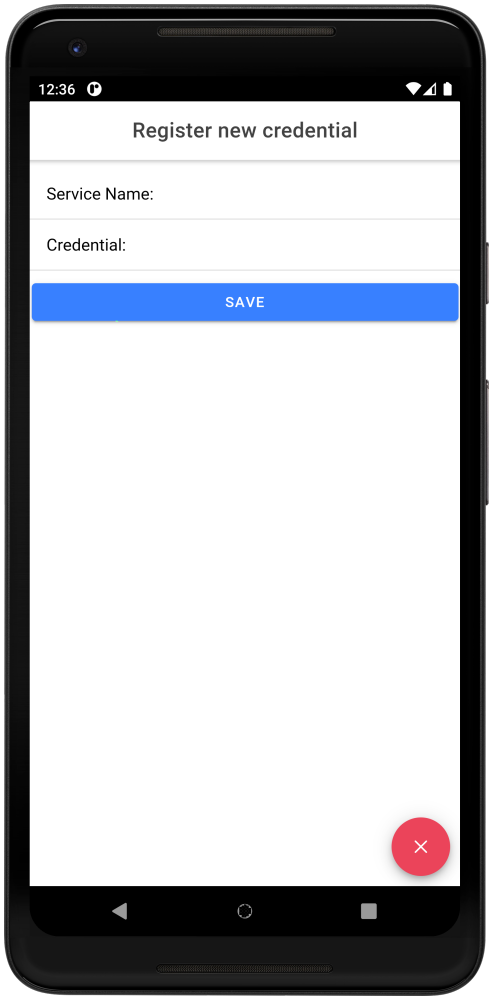

Figura 7 – Tela de cadastro do *Multifactor*

6 Conclusão

Neste capítulo apresento a conclusão deste trabalho. Inicialmente discutiremos as contribuições que mesmo para a sociedade, trabalhos futuros a serem realizados no *APP* e por fim as considerações finais.

Este projeto contribui socialmente de forma significativa pois sua licença é de código aberto além de estar disponível na plataforma *GitHub* atrevés deste link, permitindo que qualquer pessoa tenha acesso a seu código fonte o pro conta própria faça melhorias ou adaptações no mesmo, democratizando o acesso ao conteúdo.

Esta aplicação possibilita também que usuários de diversas naturezas possam incrementar a segurança de acesso a seus dados seus dados em plataformas compatíveis. O uso deste software pode auxiliar a reduzir consideravelmente os danos causados a vitimas de ataques *cyber* ataques, em especial aqueles que utilizam de engenharia social como *phishing*.

Para a continuação deste projeto é importante uma evolução da aplicação em requisitos de acessibilidade, possibilitando que portadores de deficiência tenham acesso de forma mais adequada a tecnologias de maior qualidade que sejam oriundas do ramo de segurança da informação.

Realizar a implementação de novos mecanismos de segurança para impedir que a aplicação não tenha seu conteúdo exposto caso a tela do smartfone seja fotografada ou gravada. Inserir também de leitura de informações atreves de *QRcode* e comandos de voz aumentaria de forma significativa a qualidade de vida destas pessoas no que diz respeito a proteção de seus dados e praticidade de uso de tal tecnologia.

Consuma-se que a construção de uma aplicação foi um sucesso, em diversos testes para autenticação de 2 fatores em contas de plataformas como *Google* e *Facebook* foram bem sucedidos, o que comprova a eficácia do *APP* em executar o que lhe foi proposto.

Referências

ALOUL WASSIM EL-HAJJ, S. Z. F. A. Multi factor authentication using mobile phones. International Journal of Applied Mathematics and Computer Science, v. 2, p. 65–80, 2009. ISSN 1814-0424. Disponível em: <https://www.researchgate.net/publication/228972704_Multi_Factor_Authentication_Using_Mobile_Phones>. Citado na página 10.

APWG. *Phishing Activity Trends Report, 2nd Quarter 2020*. [S.l.], 2020. Disponível em: <https://docs.apwg.org//reports/apwg_trends_report_q2_2020.pdf>. Citado na página 11.

BASIT A., Z. M. L.-X. e. a. A comprehensive survey of ai-enabled phishing attacks detection techniques. *Telecommun Syst*, v. 76, p. 139–154, January 2021. ISSN 1572-9451. Disponível em: <https://doi.org/10.1007/s11235-020-00733-2>. Citado na página 10.

LAMPORT, A. I. . C. L. Password authentication with insecure communication. Communications of the ACM, v. 24, p. 770–772, 1981. Disponível em: <https://doi.org/10.1145/358790.358797>. Citado na página 10.

RABOY, N. *Phishing Activity Trends Report, 2nd Quarter 2020*. [S.l.], 2014. Disponível em: <https://www.thepolyglotdeveloper.com/2014/10/generate-time-based-one-time-passwords-javascript/>. Citado na página 11.

RABOY, N. *Build A Time-Based One-Time Password Manager With Ionic 2*. [S.l.], 2016. Disponível em: <https://www.thepolyglotdeveloper.com/2016/08/build-time-based-one-time-password-manager-ionic-2/>. Citado na página 11.

VERIZON. *2021 Data Breach Investigations Report*. [S.l.], 2021. Disponível em: <https://www.verizon.com/business/resources/reports/2021-data-breach-investigations-report.pdf>. Citado na página 10.