UNIVERSIDADE FEDERAL DO PARANÁ

HUGO HENRIQUE CAMARA COSTA
RICARDO TOSIN

VIZINHO SOLIDÁRIO
UM APLICATIVO DE SEGURANÇA COMPARTILHADA

CURITIBA
2016

HUGO HENRIQUE CAMARA COSTA
RICARDO TOSIN

Vizinho Solidário
Um Aplicativo de Segurança Compartilhada

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal do Paraná como requisito para obtenção do diploma.

Orientador: Prof. Setembrino Soares Ferreira Jr.

Curitiba - PR
Dezembro/2016

RESUMO

Uma das maiores evoluções humanas se proveu da comunicação. Desde os tempos primitivos a comunicação era o diferencial para alimentação e segurança dos homens antigos, ditos homens das cavernas. Com a evolução desta ferramenta humana e digital chamada comunicação e com a chegada do telefone e da *internet*, juntamente com a evolução dos computadores para dispositivos totalmente móveis, a maneira de comunicação se tornou algo ágil e eficaz. O objetivo deste trabalho é prover uma ferramenta que, em tempos modernos, chamamos de aplicativos para dispositivos móveis; este seria um *App,* capaz de cadastrar um grupo de uma determinada localização, como uma rua de uma cidade, onde seus moradores podem ter um tipo de comunicação eficaz, enviando alertas de segurança, entre outras informações primordiais, para manter a ordem em uma sociedade que visa à civilidade e segurança compartilhada de seus patrimônios. Este trabalho foi desenvolvido utilizando ferramentas open source e plataformas para o desenvolvimento de *Apps* híbridos, que visam a multiplataforma de sistemas através de uma única criação. Iremos verificar se este tipo de metodologia substitui o aplicativo nativo no desenvolvimento de um *App* de segurança compartilhada. Também será verificado se há aceitação do *App* na sociedade e se há espaço para o mesmo entre os aplicativos de redes sociais e conversas em grupo. Somado há isso, analisaremos a complexidade de desenvolvimento, visto que sua execução transcende o programa de celular, pois há a necessidade de um banco de dados, servidor *Web*, ou seja, é necessária a integralização desses sistemas para o funcionamento completo do *App* em questão.

.

Palavras-chave: *App*; Dispositivos Móveis;

ABSTRACT

One of the biggest evolution in the human race became true with the communication, the messages between the humans since the cave age where the men's called “men of the cave”, start to communicate with signals, the evolution of this human and digital tool called communication and with an arrival of the telephone and the Internet Together with an evolution of computers to mobile devices a way of communication has become something agile and effective, the purpose of this work is to prove a tool in modern times we call mobile applications, this is an application, able to register a group of a fixed location such as a city street, where its residents can have an effective type of communication, sending security alerts among other key information to maintain order in a society that aims at a civility and shared security of their assets. This work was developed using open source tools and platforms for the development of hybrid Apps, which aim at multiplatform systems through a single creation. We will check if this type of methodology replaces the native application in the development of a shared security app. It will also be verified if there is acceptance of the App in society and whether there is room for the same between social networking applications and group conversations. Added to this, we will analyze the development complexity, since its execution transcends the mobile program, since there is a need for a database, Web server, that is, it is necessary to integrate these systems for the complete operation of the App in question.

Keywords: App; Mobile Devices;

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E/OU SIGLAS

APP - Aplicativo de dispositivo Móvel (*APPlication*)
API - *Application Program Interface*
SDK - *Software Development Kit*
CMS - *Content Management System*
REST - *REpresentational State Transfer*
HTML - *HyperText Mark-up Language*
DOM - *Document Object Model*
HTTP - *Hypertext Transfer Protocol*
MVC - *Model-View-Controller*
JSON - *JavaScript Object* Notation
CSS - *Cascading Style Sheets*
DI - *Dependency Injection*
JSON - *JavaScript Object Notation*

Sumário

1. INTRODUÇÃO

O telefone foi uma das grandes invenções da humanidade, que revolucionou a comunicação, permitindo que fosse feita através de grandes distâncias. Os dispositivos móveis modernos são primos distantes dos telefones antigos, com eles podemos mandar mensagens de voz e texto, realizar pagamentos sem abrir a nossa carteira e, claro, fazer ligações como antigamente, entre outras milhares de funcionalidades [1].

Hoje os dispositivos móveis são vistos mais como computadores do que propriamente telefones, são conhecidos como *smartphones*. Começaram a surgir em meados de 2002, com teclados QWERTY, telas *touch*, acesso à *internet*, entre outras funcionalidades. Com essa evolução surgiram novas tendências, a criação de aplicativos específicos para dispositivos móveis, e tal fato também mudou o comportamento da *internet*, onde hoje existem *sites* divididos para funcionar em *desktops* e em dispositivos móveis [1].

Desde então, diversas ferramentas como *frameworks* foram desenvolvidos para os diferentes sistemas operacionais, que são chamados de ferramentas para desenvolver aplicativos nativos, ou seja, aplicativos que só irão funcionar para o seu modelo de sistema específico, por exemplo, para o *Android* temos o *Android Studio* e para o *iOS* o *XCODE*. Porém existem também *frameworks* para ambos os sistemas, como no caso para os aplicativos híbridos.

- Aplicativos Nativos: São geralmente desenvolvidos utilizando linguagens de alto nível, como Java no *Android*, *Objective-C* no *iOS*, ou C# no *Windows Phone*. As *APIs* nativas são disponibilizadas para os desenvolvedores como parte da plataforma *SDK*. Essas plataformas são desenvolvidas para fornecerem acesso otimizado para as capacidades do *hardware* para o aplicativo nativo. Em contrapartida, é necessário desenvolver o aplicativo em cada plataforma *SDK* para deixá-lo “multiplataforma”. Aplicativos nativos são indicados quando a

otimização do desempenho é crítica, por exemplo, em simulações e interações gráficas *high-end*. [11]

- Aplicativos Híbridos: É uma categoria especial de aplicativos *Web,* que estendem o ambiente dos aplicativos base *Web* através do uso de APIs das plataformas nativas disponíveis em um determinado dispositivo. O padrão de *design* de um aplicativo híbrido é igualmente aplicável em ambos os ambientes, móveis e *desktop.* [11]

1.1 Objetivo Do Trabalho

Este trabalho tem como objetivo o desenvolvimento prático de um aplicativo híbrido para dispositivos móveis, capaz de oferecer segurança residencial compartilhada, com foco em compartilhamento de informações entre usuários. Onde os usuários poderão compartilhar situações atuais através do aplicativo e informar seus vizinhos.

1.2 Organização Do Documento

O documento inicia apresentando os conceitos necessários para a compreensão do trabalho, assim como as tecnologias envolvidas na implementação do mesmo. Apresentados os conceitos, então é descrito o planejamento da implementação, onde foram tomadas decisões quanto às tecnologias e os detalhes de implementação. Em seguida, é descrito em um modelo “passo a passo” como a aplicação realiza a tarefa proposta: desenvolver um aplicativo para segurança compartilhada.

2. CONCEITOS INICIAIS

Este capítulo contempla os conceitos básicos envolvidos nesse trabalho. O objetivo do capítulo é servir como uma base para a compreensão do que foi feito e o porquê das decisões tomadas, estas descritas no capítulo seguinte, quanto às tecnologias e métodos envolvidos.

2.1 Modelo Do Aplicativo

Para o desenvolvimento e protótipo do aplicativo, optou-se por utilizar ferramentas e ou *softwares* "*open source*" gratuitos, permitindo assim ter um custo mínimo de desenvolvimento de código e base de dados, visando atingir uma gama de público maior onde, além do sistema *Android* pode-se atingir sistemas como *Windows Phone* e *iOS* [8].

Neste modelo de aplicativo híbrido, temos a possibilidade de desenvolver o *App* através de linguagens como *Java Script*, utilizando o *framework AngularJS*, nosso *Front End,* facilitando o desenvolvimento, pois integra *HTML* com o *Java Script* de uma maneira única; e então, para melhorar ainda mais a aplicação e sua integração com os sistemas operacionais móveis, iremos utilizar o *framework* conhecido como *Ionic* e integrar com o *CMS Wordpress* através de uma *Rest API*, o *Back End*, utilizando um padrão de desenvolvimento conhecido como *MVC*, que já é aplicado originalmente no *AngularJS*.

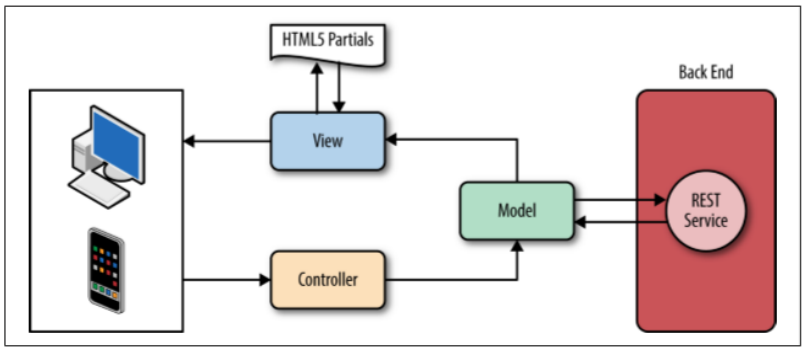

Figura 1: Diagrama *MVC* aplicação *AngularJS*.

2.2 *HTML DOM*

O *DOM* (*Document Object Model*) é uma interface de programação de aplicativos (*API*). Define a estrutura lógica dos documentos e o jeito que são acessados e manipulados. Com o *DOM*, programadores podem criar documentos, navegar nas suas estruturas, e adicionar, modificar, ou apagar elementos e conteúdos. Qualquer informação encontrada em um documento *HTML* pode ser acessada, alterada, apagada ou adicionada utilizando Modelo de Objeto de Documento (*DOM*) [5].

2.3 *Scripts*

Scripts são pequenos programas, transparentes para o usuário, e vão nos "bastidores" de ajuste fino das páginas *Web*. Os próprios *Scripts* podem ter uma entre várias funções. A função clássica é renderizar a forma para parecer "inteligente" para interagir com o usuário enquanto ele preenche. Então, pode-se usar um script para cada campo de um formulário para garantir que seja preenchido de forma correta, e então avisar caso tenha deixado para fora alguma informação importante para o mesmo [2].

2.4 *Java Script*

Java Script é uma linguagem baseada em texto que não precisa de conversão antes de ser executada. Outras linguagens, como o *Java*, precisam ser compiladas para se tornarem executáveis, mas Java Script é executada instantaneamente por um tipo de programa que interpreta o código, chamado de analisador ("*parser*"). Quase todos os navegadores Web possuem um analisador Java Script.

Há duas formas de se executar *Java Script* no navegador. A primeira seria colocá-lo dentro de um elemento *script* em qualquer lugar de um documento *HTML*. A outra forma seria colocar em um arquivo *Java Script* e então referenciá-lo dentro do documento *HTML* usando um *script* vazio com o atributo *src* [5].

2.5 *AngularJS*

AngularJS é um framework estrutural para aplicativos *Web* dinâmicos. Proporciona o uso do *HTML* como linguagem modelo e com a extensão de sua sintaxe para expressar de forma clara e sucinta os componentes da aplicação [5].

2.6 Padrão *MVC*

O padrão arquitetural *MVC* (*Model View Controller*) separa uma aplicação em três grupos principais: *Models*, *Views* e *Controllers.* Esse padrão ajuda a separar as preocupações. As requisições do usuário são enviadas para o *Controller* que é responsável por trabalhar com o *Model* para desempenhar as ações do usuário e/ou retornar os resultados das consultas. O *Controller* escolhe qual *View* irá mostrar para o usuário, e a fornece com qualquer dado do *Model* requerido. [3].

2.7 *JSON*

JSON (*Java Script Object Notation*) é um formato de dados muito leve e extremamente fácil para escrita e leitura de um humano, baseado em subconjuntos de linguagem *Java Script*, é extremamente útil para a máquina analisar e gerar resultados. Apesar de ser uma linguagem completamente independente, usa convenções familiares com outras linguagens como *C*, *C++* e *JAVA*[7].

2.8 *IONIC*

Ionic é uma plataforma em *HTML5* de desenvolvimento de *App* para celulares. Tem como objetivo a criação de aplicativos híbridos (multiplataforma). *Apps* híbridos são essencialmente pequenos *websites* rodando em um *shell* do navegador, em um *App* que tem acesso à camada da plataforma nativa. Ionic é uma biblioteca *Java Script* de interface do usuário. Vários componentes são nativos do *Java Script* para poderem rodar adequadamente, entretanto

geralmente componentes podem ser usados sem codificação pelas extensões da plataforma, como a extensão Angular Ionic [8].

2.9 *WORDPRESS*

Wordpress é um *CMS* (*Content Management System*) *open source*, usado para criar e manter conteúdo digital na internet. Com ele é possível desenvolver uma diversidade de modelos de blogs, sites, lojas virtuais, redes sociais e até mesmo aplicativos para dispositivos móveis. Utiliza como base um banco de dados *MYSQL* e linguagens *Php* e *Java Script*, entre outras linguagens básicas na *Web*, como *CSS HTML* [6]. Hoje representa 58% da fatia do mercado *Web* [9], uma grande fatia da internet mundial; cerca de milhões de sites depende do Wordpress [9].

2.10 *CSS*

CSS (*Cascading Style Sheets*) é a linguagem para descrever a apresentação de páginas *Web*, incluindo cores, modelo, e fontes. Ela permite adaptar a apresentação para diferentes tipos de dispositivos, como em telas grandes e pequenas. *CSS* é independente ao *HTML* e pode ser usado com qualquer linguagem de remarcação da base *XML*. A separação do *HTML* para o *CSS* deixa a manutenção fácil de *sites*, compartilhar estilo entre páginas, e adaptar páginas a diferentes ambientes [5].

2.11 *REST API*

API (*Application Program Interface*) é a padronização de jeitos que um pedaço particular pode ser usado. As regras são definidas para a sua interação com o resto, as quais governam como outros pedaços do software podem conversar com o programa e como irão responder. O *REST* (*Representational State Transfer*) *API* é um tipo específico arquitetural de abordagem para concatenar essas regras. É uma metodologia projetada para permitir que programas conversem entre si da forma mais simples possível [10].

3. CONCEITOS ESSENCIAIS DO APLICATIVO

Iremos introduzir alguns conceitos essenciais para o entendimento do desenvolvimento do aplicativo; estes conceitos envolvem diretamente o framework *AngularJS*.

As Aplicações desenvolvidas com *AngularJS* normalmente são chamadas de *Single Page Applications (*aplicações de páginas únicas), onde normalmente temos uma página *HTML*, e todas as aplicações são carregadas dinamicamente dentro da página. Outro conceito muito importante é o *Bootstrapping,* onde ao iniciar o carregamento das páginas, imediatamente é carregada a biblioteca do AngularJS dentro das *tags HTML* [12].

Uma das principais características do AngularJS é o uso do padrão de design conhecido como *Dependency Injection* (DI). Este padrão é definido como parte da configuração da aplicação que, ao carregar o código, no início da aplicação, já consegue definir e carregar as dependências da mesma, o que favorece o teste da aplicação e, principalmente, evita o problema de *boilerplate code*, onde há a transcrição repetida de código com alterações mínimas [4].

3.1 Módulos

Módulos são sem dúvidas uma das principais definições dentro da aplicação. Podemos dizer que a aplicação é dividida em módulos. Existem diferentes tipos de módulos. São eles que definem como será o *bootstrap* da aplicação. Não existe um método principal, como na maioria das aplicações, mas sim módulos que dão uma vantagem em questão ao entendimento do código, no reuso e teste, separando em partes os desenvolvimentos [12].

3.2 Controllers

Controllers são uma das principais diretivas no *AngularJS*, são controladores que definem a manipulação do *DOM* no *HTML* e devem conter a Lógica do Negócio, através de um conceito chamado de *data binding*, onde qualquer alteração no *model* reflete diretamente na alteração da *View*[9].

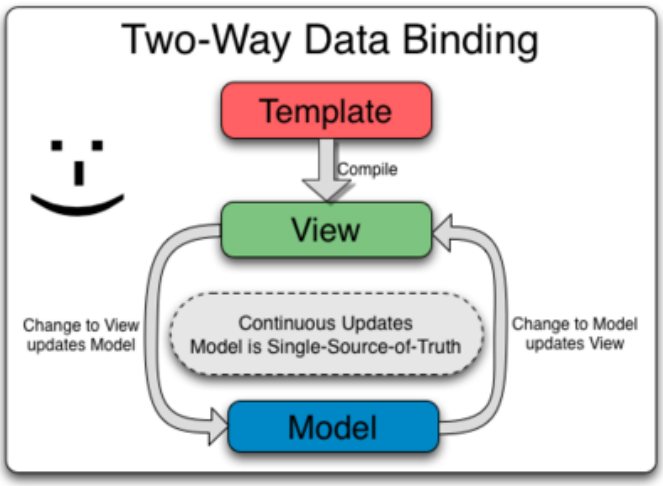

Figura 2: *Data Binding* in *AngularJS*

3.2.1 Services

Services são objetos de serviços associados às diretivas. Como filtros controladores, eles estão diretamente ligados ao padrão (DI), fazem uma instanciação preguiçosa, ou seja, só são utilizados quando a aplicação tem dependência dos mesmos. Além disso, são *Singletons,* onde cada componente dependente deste serviço recebe uma única instância do mesmo [9].

4. PESQUISA DE VIABILIDADE DO APP

Para verificar a viabilidade de aceitação do aplicativo foi realizada uma pesquisa com os seus pontos chaves. A pesquisa foi feita de forma *online,* utilizando o *Google Forms*. Ela ficou aberta para o público pelo período de um (1) mês, de 18/10/2016 até 18/11/2016. A pesquisa abrangeu um total de 188 pessoas residentes à cidade de Curitiba.

4.1 Gráficos das perguntas feitas

4.1.1 Quantas conhecem os seus vizinhos?

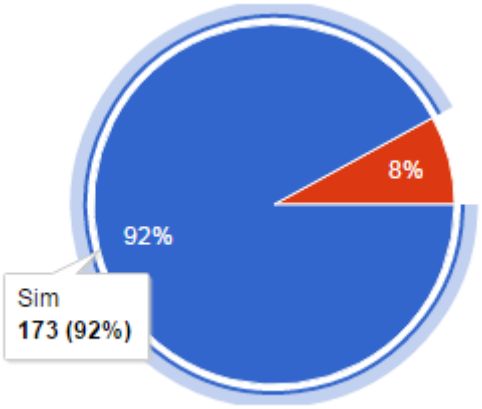

Figura 3: Pessoas que conhecem seus vizinhos

Na Figura 3 é possível verificar que quase todas as pessoas (92%), que responderam o questionário, conhecem pelo menos um vizinho.

4.1.2 Quantos vizinhos conhecem?

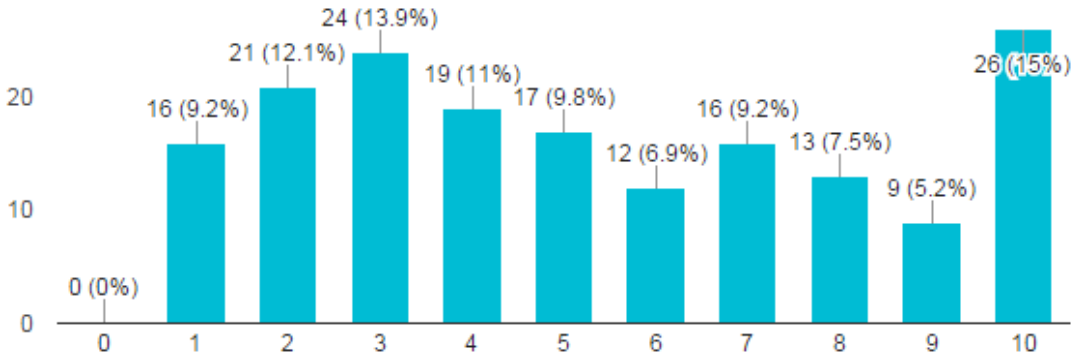

Figura 4: Quantos vizinhos conhecem?

Em relação às pessoas que responderam que conhecem alguns vizinhos, a pergunta de número dois questionava a quantidade de vizinhos que é conhecido. Utilizando uma escala de 0 a 10, sendo zero igual a nenhum vizinho e 10 todos. Pode-se concluir que a maioria dos entrevistados conhece, pelo menos, a metade dos seus vizinhos, como pode ser visto da Figura 4.

4.1.3 Quantas pessoas possuem um grupo de conversa com os vizinhos?

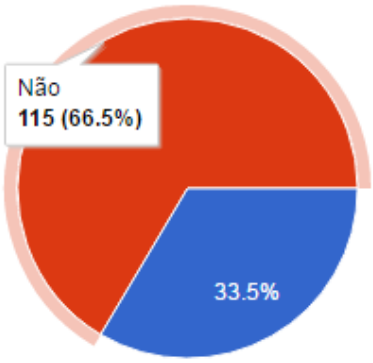

Figura 5: Quantas pessoas possuem grupo com vizinhos?

Continuando no grupo de pessoas que conhecem seus vizinhos, a pesquisava continuava perguntando se a pessoa possuía um grupo de conversa ou outro artifício para se comunicar com seus vizinhos. Na figura 5 é possível verificar que mais de 66% das pessoas não se comunicam com seus vizinhos.

4.1.4 Como as pessoas se comunicam com seus vizinhos?

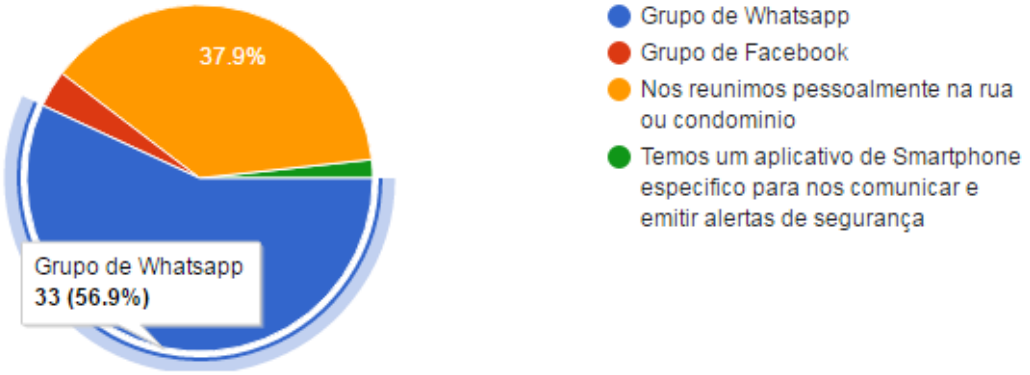

Figura 6: Como se comunicam com seus vizinhos?

Dentre as pessoas que possuem um tipo de comunicação com os vizinhos, a pergunta 4 questionava qual tipo este seria. Sendo os mais respondidos: grupo no *Whatsapp* e reuniões na rua ou condomínio, como pode ser visto na figura 6.

4.1.5 Quantas pessoas já tiveram a casa assaltada?

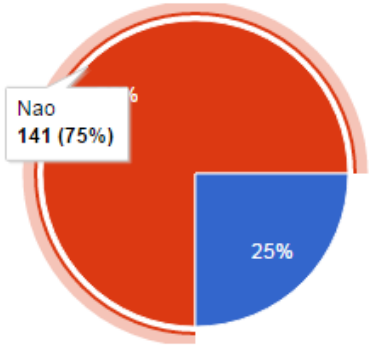

Figura 7: Quantas pessoas já tiveram a casa assaltada?

A figura 7 demonstra em forma de gráfico quantas pessoas já tiveram a sua casa assaltada.

4.1.6 Como ficaram sabendo do assalto?

A partir das pessoas que responderam que sua casa já foi assaltada, a figura 8 mostra a forma como essas pessoas ficaram sabendo do assalto. E que o maior grupo só ficou sabendo na hora em que chegou a casa e viu tudo revirado.

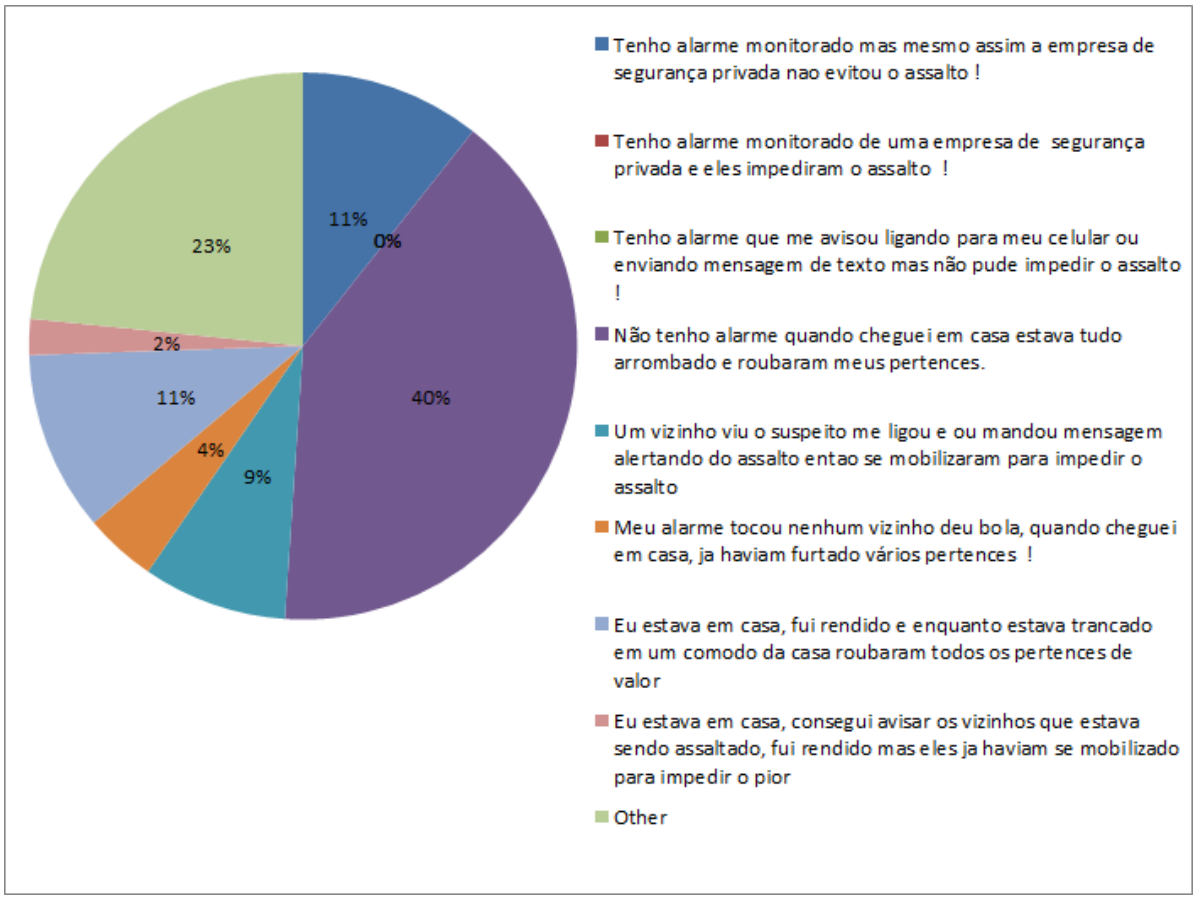

Figura 8: Como ficaram sabendo do assalto?

4.1.7 Quantas já souberam de um assalto às residências vizinhas?

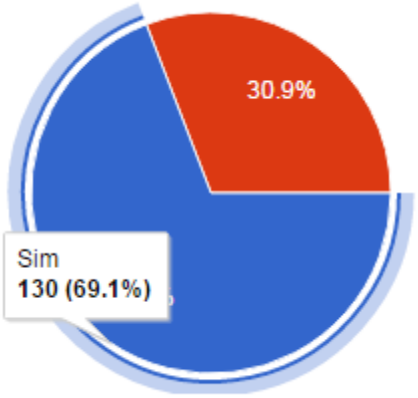

Figura 9: Quantas souberam de assalto ao vizinho ?

A pergunta de Figura 9 demonstra a quantidade de pessoas que já souberam de um assalto ao vizinho.

4.1.8 Quantas já presenciaram um assalto às residências vizinhas?

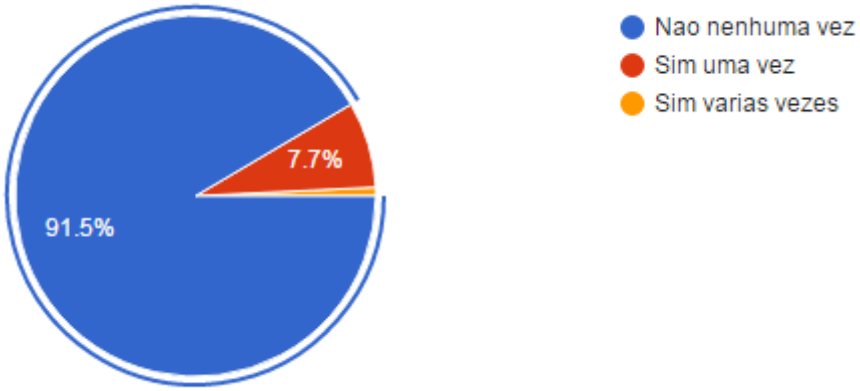

Figura 10: Quantas pessoas presenciaram assalto ao vizinho?

Em relação às pessoas que tiveram ciência do assalto ao vizinho, quantas delas presenciaram o fato. Como pode ser visto na Figura 10, poucas estavam presentes na hora do assalto.

4.1.9 Como reagiram?

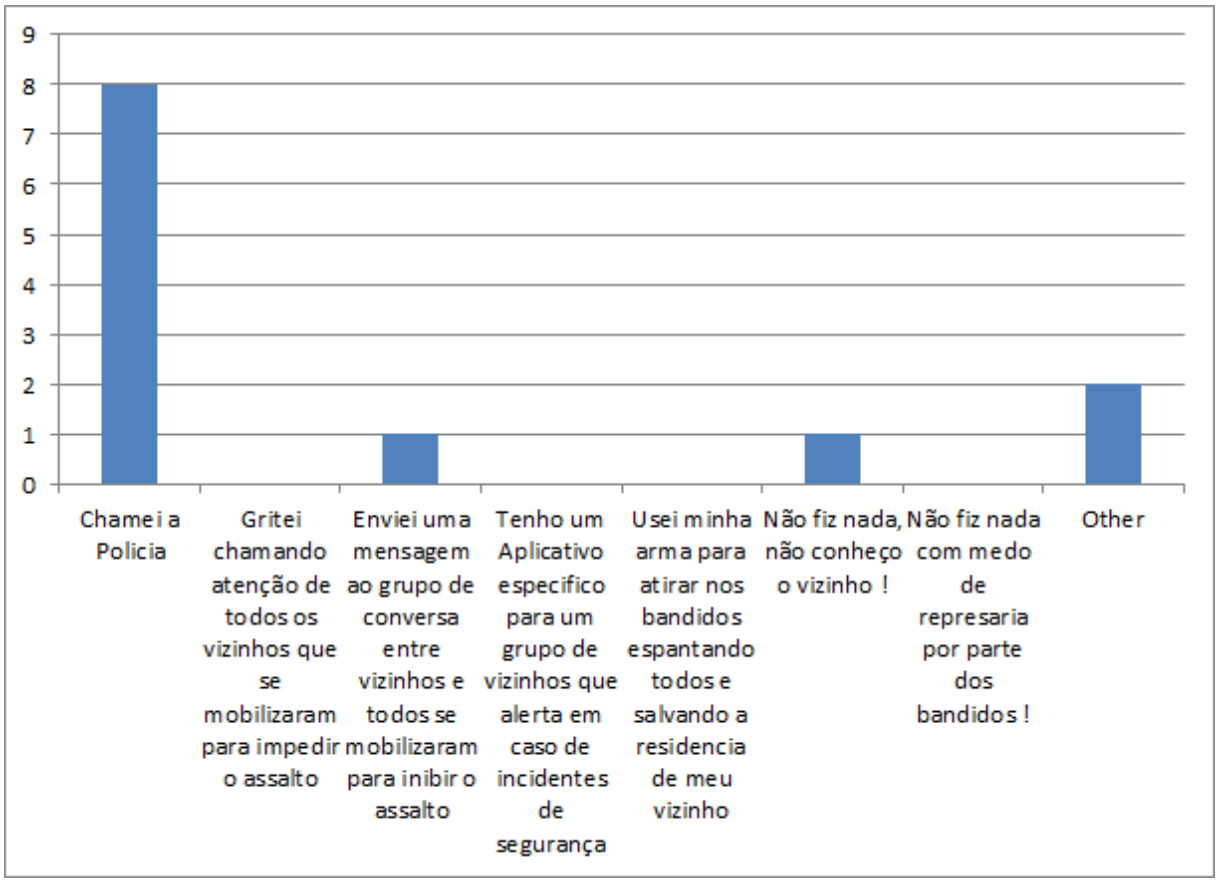

Figura 11: Como reagiram ao assalto ao vizinho?

Dentre as pessoas que estavam quando o assalto acontecia, a questão 9 pergunta qual foi a atitude tomada pela pessoa. Sendo este, como pode ser visto na Figura 11, foi chamar a polícia.

4.1.10 Quantas usariam um *App* de segurança compartilhada que emite alerta de incidentes

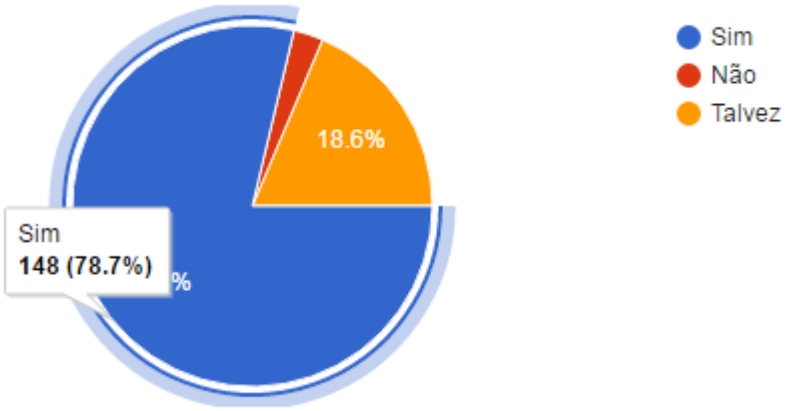

Figura 12: Gráfico usariam A*pp* de segurança

A figura 12 demonstra, em forma de gráfico, o resultado obtido pelas respostas da pergunta 10, que indagava as pessoas se elas usariam um aplicativo para celular de segurança compartilhada, onde nele é possível emitir alerta de incidentes de segurança para os seus vizinhos.

4.2 Conclusão da pesquisa

Através dos dados coletados foi possível verificar a aceitação positiva do aplicativo pelas pessoas entrevistadas. Visto que 78,7% (148 pessoas) dos entrevistados usariam o *App*, 18,6% (35 pessoas) talvez usassem e somente 2,7% (5 pessoas) não possuem interesse.

5. PLANEJAMENTO

Considerando os conceitos dissertados acima, este capítulo busca resumidamente explicar o funcionamento conceitual do aplicativo, mostrando um breve caso de uso, como foi feito o desenvolvimento do *Front End* e *Back End,* entre outras funcionalidades específicas, para que possa ser desenvolvido um aplicativo de segurança compartilhada, e até mesmo outros modelos de aplicativo.

5.1 Funcionamento e Casos De Uso

Iremos descrever o funcionamento do aplicativo na interação usuário e aplicativo, e então iremos demonstrar um caso de uso para os alertas de segurança e outras opções que consideramos mais importantes e específicas no funcionamento do aplicativo. Será permitido ao usuário somente estar cadastrado em um grupo, sendo este administrador ou não.

5.2 Descrevendo o Funcionamento

Após encontrar e efetuar o download do aplicativo Vizinho Solidário nas lojas de aplicativos, o usuário executa o aplicativo, O aplicativo mostra a tela de boas vindas ao usuário, com uma pequena introdução explicando o seu funcionamento básico. Então o App irá verificar se é a primeira interação do usuário, com o mesmo solicitando o cadastro ou lendo informações de cadastro salvas no aparelho.
Caso o usuário não seja cadastrado, este irá efetuar um cadastro através de e mail, *Facebook* ou conta *Google*: as informações necessárias são nome, sobrenome, e-mail e número de telefone.

Após o cadastro, são apresentadas duas opções ao usuário: criar grupo e entrar em grupo com código. Caso selecione criar grupo, o usuário será o administrador do grupo criado, então são apresentadas as opções, o usuário

deve cadastrar um alias, ou seja, um apelido para aquele grupo, que pode ser um nome, como “Rua Peru” ou Condomínio Torres, com o propósito de ser o lugar onde serão compartilhadas as informações de segurança, assim como informações como cidade e bairro. Então é solicitado o cadastro pessoal da propriedade do usuário, como Nome do Usuário no Grupo, Rua, número da casa, um alias ou apelido de identificação; exemplos: ”Sobrado 21”, ou “Casa Laranja”, que irá pertencer àquele grupo.

Caso a seleção seja inserir código de grupo o usuário insere o código no formulário, e este irá encaminhar o usuário para o cadastro, que é o mesmo cadastro da segunda etapa de Criar Grupo.

5.3 Caso de uso Emitir Alerta

1) **Nome do Caso de Uso**
    - **a.** Emitir Alerta
2) **Breve Descrição**
    - **a.** Emitir um alerta, selecionando o tipo de alerta, comigo ou com o vizinho. Assim seleciona qual o alerta dentre os eventos assalto, alarme, incêndio ou portão aberto.
3) **Fluxo de Eventos**
    - **a.** Fluxo Básico
        - **i.** O caso de uso começa com o ator selecionando a opção emitir alerta; o sistema apresenta uma tela com os campos:
            1. “Alerta Comigo”
            2. “Alerta com Vizinho”.
        - **ii.** Caso o ator selecione a opção “Alerta Comigo”, o sistema mostra uma tela com as opções Alertar Assalto, Alertar Incêndio, Alertar Médico, Alertar Portão Aberto.
        - **iii.** O ator seleciona uma das opções de alerta, e o sistema notifica o usuário com a mensagem:
            1. “Alerta Enviado”.

iv. Caso o ator selecione a opção “Alerta com Vizinho”, o sistema mostra a tela com as opções Alertar Assalto, Alertar Incêndio, Alertar Médico ou Alertar Portão Aberto.

v. O ator seleciona uma das opções de alerta, e o sistema mostra uma tela com a lista de vizinhos cadastrados nos sistemas.

vi. O ator seleciona um dos vizinhos na lista e o sistema notifica o ator com a mensagem “Alerta Enviado”.

4) Requisitos especiais

a. Não se aplica.

5) Pré-condições

a. Necessário estar logado e cadastrado no sistema.

b. Qualquer perfil de usuário é válido.

6) Pós-condições

a. Ter selecionado um tipo de alerta.

7) Pontos de extensão

a. Não se aplica.

5.4 Diagrama de Caso de Uso

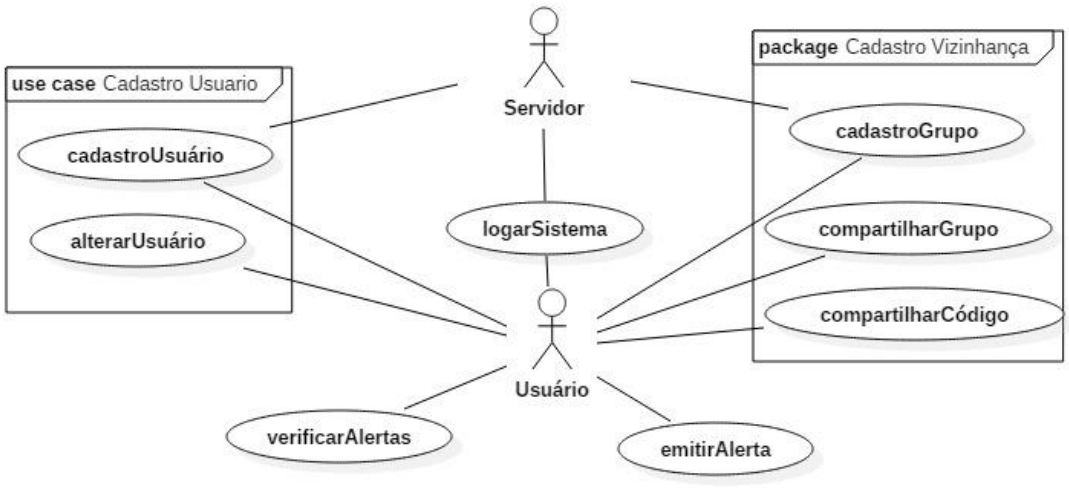

Figura 13: Diagrama Caso de Uso do App

5.5 Diagrama de Sequência Cadastrar Vizinhança

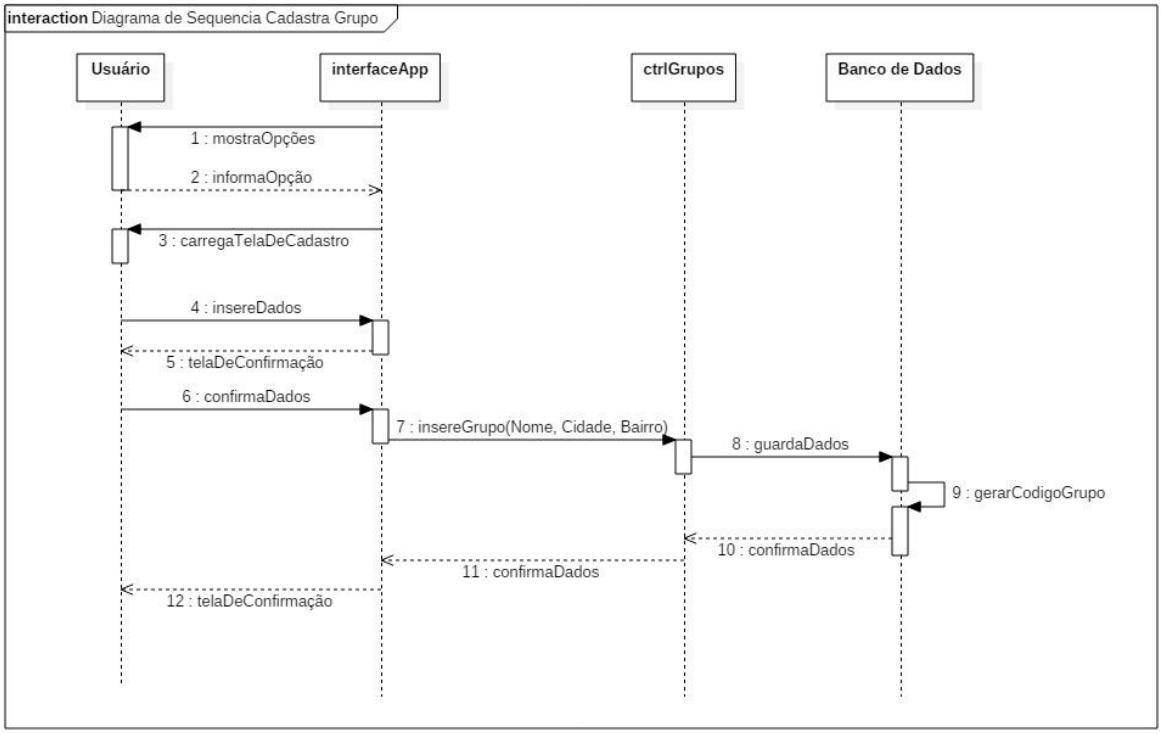

Figura 14: Diagrama de Sequência Cadastrar Grupo

5.6 Diagrama de Sequência Emitir Alerta

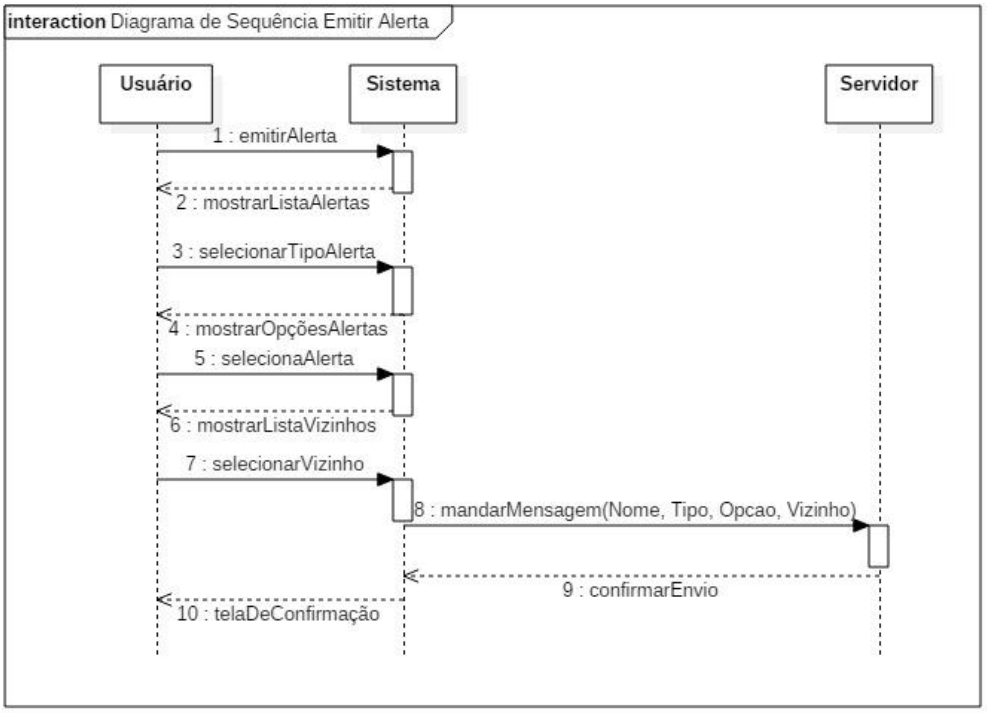

Figura 15: Diagrama de Sequência Emitir Alerta

5.7 Interface Com o Usuário

Para o desenvolvimento da interface gráfica, foi utilizada a biblioteca online do Ionic, *Ionic Creator* - [6]. Esta permitiu a construção de uma aplicação gráfica em *HTML* 5 com *Java Script* e *CSS*.

5.8 Protótipo Inicial

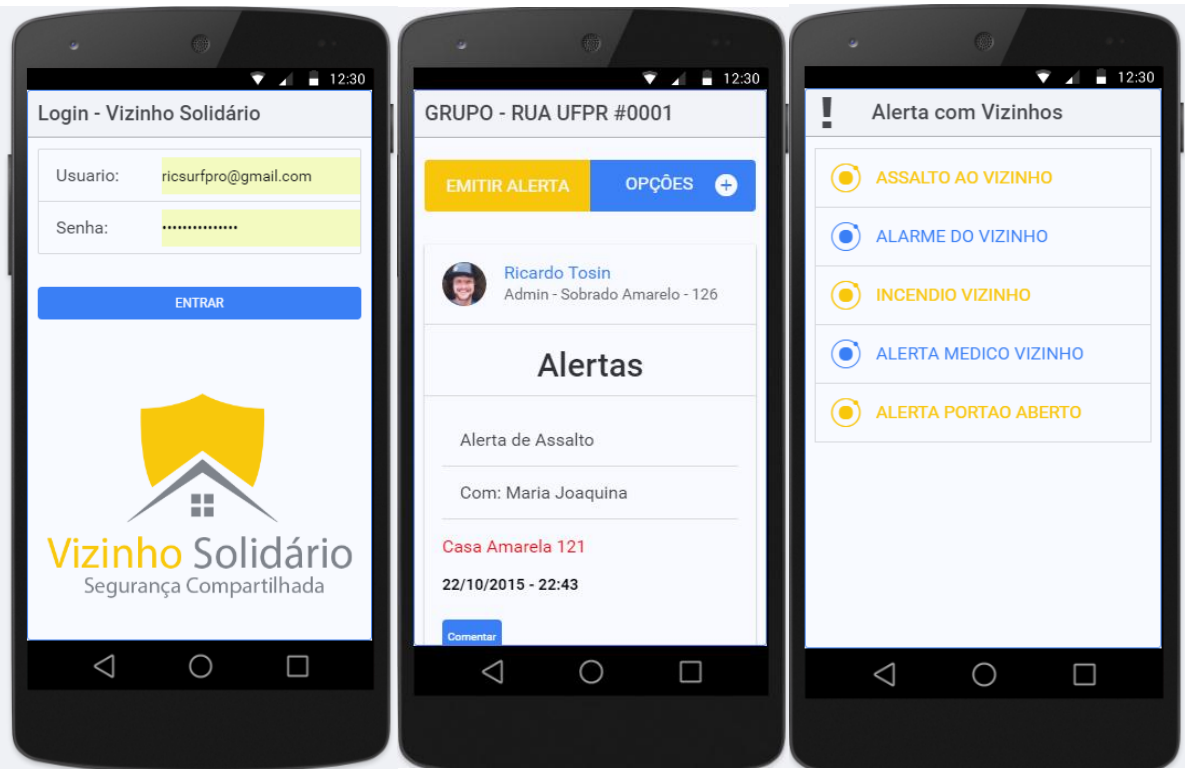

Figura 16: Interface Gráfica *APP*

5.9 Desenvolvendo o *Front End*

O desenvolvimento da aplicação iniciou-se com a verificação das propriedades de "*Routes*" do modelo. *Routes* são, basicamente, as rotas que definem qual página será carregada no aplicativo, por exemplo: se o usuário clicar nas opções de Emitir Alerta, este deverá carregar a página para que o usuário selecione o tipo de alerta. O modelo construído com o criador de *layouts* do

Ionic nos permite partir com esta função já estabelecida e codificada no arquivo "*routes.js*"; após isso deu-se inicio às funções que implementam os "*Services*" do *AngularJS*, basicamente os serviços são para conexão com a *REST API* e *REST API JSON* através do *HTTP*, para executar os métodos *get* e *post*. A escolha deste método foi feita tendo em mente dois pontos. Primeiramente, a facilidade em implementá-lo e a necessidade de logar o usuário no sistema para interagir com o mesmo. E em segundo lugar, para manter o propósito de padrão de desenvolvimento, ao fazer uma solicitação à *API*, não recebemos uma resposta imediata, mas sim uma promessa de que haverá uma resposta, em formato objeto *Json*.

Para fazer o *login* iremos utilizar o método de autenticação através de "*cookies*", não iremos nos preocupar com a segurança visto que estamos desenvolvendo a ideia de um protótipo. Primeiramente enviamos uma requisição para *API* através do método *get,* solicitando um *cookie* válido, passando o usuário e senha pelo header do *HTTP*. Como não temos *HTTPS* neste *site,* o método é totalmente inseguro para passar o usuário e a senha; a *API* nos retorna um cookie válido, para que possamos solicitar a autenticação e então solicitamos à *API* que autentique o *cookie,* como mostramos abaixo.

5.10 Exemplo do Uso da *JSON API*

5.10.1 Solicitação de *Cookie* de Autenticação

5.10.1.1 Solicitação pelo Método *HTTP*

http://www.tosinhost.com/vizin/api/auth/generate_auth_cookie/?insecure=cool&username=maria&password=mariamaria

5.10.1.2 Resposta em objeto *JSON*

{"status":"ok","cookie":"maria|1482958977|mvfBgjwONAkCPQf2sFCWHzAWdOWeS7MEW8Bm5jk2jYN|0ae2e760c0e14f3dd942f6520662daea612a18e28d46f5a3a3e83cd40c2e6cb5"}

5.10.2 Validação do *cookie*

http://www.tosinhost.com/vizin/api/auth/validate_auth_cookie/?insecure=cool&cookie=maria|1482958977|mvfBgjwONAkCPQf2sFCWHzAWdOWeS7MEW8Bm5jk2jYN|0ae2e760c0e14f3dd942f6520662daea612a18e28d46f5a3a3e83cd40c2e6cb5

5.10.2.1 Resposta em objeto *JSON*

True.

5.10.3 Emitindo alertas através da *API*

Para o usuário logado emitir um alerta, primeiramente devemos solicitar à *API* um *nonce*, este é um número único gerado para somente permitir que nenhuma conexão antiga possa se sobrepor a esta conexão; então, após isso colocamos o *nonce* no *header* da mensagem, juntamente com o que desejamos como título do *post,* categoria e subcategoria, entre outras informações que serão utilizadas no *App* no caso da programação.

5.10.4 Gerando *Nonce* através da *API*

http://www.tosinhost.com/vizin/api/get_nonce/?controller=posts&method=create_post

5.10.4.1 Resposta

{"status":"ok","controller":"posts","method":"create_post","nonce":"432b62eb69"}

5.10.5 Fazendo uma postagem de alerta na *API*

http://www.tosinhost.com/vizin/api/posts/create_post/?nonce=d4720ae783&&title=Assalto%20Teste&status=publish&categories=alertas,assalto_comigo

5.10.5.1 Resposta

São todas as postagens em formato objeto com a última postagem realizada.

5.10.6 Carregando os Alertas no *App*

Para carregar um alerta no aplicativo e mostrá-lo em tela, vamos demonstrar uma pequena linha de código do *AngularJs* para que fique explícito o funcionamento e que fique claro que este método é utilizado para as outras funcionalidades.

```
angular.module('VizinSolidarioWordpressApp')
.factory('wpFactory', function ($http, $q) {

  var url = 'http://www.tosinhost.com/vizin/wp-json/wp/v2/';

  function getPosts(numpost) {
    return ($http.get(url+'posts?per_page=' + numpost)
    .then(handleSuccess, handleError)); //Promessa da requisição
}

  function handleSuccess(response) {
    return response.data;//Resposta objeto
}

  function handleError(response) {
    if (!angular.isObject(response.data) || !response.data.message) {
      return($q.reject("Erro Ocorrido."));
    }
    return($q.reject(response.data.message));
}

return({
    getPosts: getPosts
  });
});
```

Figura 17: Funcionamento da *Factory*

A imagem do código acima demonstra como funciona a *Factory* para gerar o serviço que espera a promessa de resposta; quando a resposta chega fica no objeto que está em "*response*.*data*", então é retornado pela *Factory*.

5.11 Desenvolvendo o *Back End* Do Aplicativo

O desenvolvimento do *back end* da aplicação é extremamente simples. Iniciou-se com a hospedagem de um sistema *Wordpress*, que irá armazenar todos os dados de usuário e postagens. O *Wordpress* possui a opção de instalar uma diversidade de *plug-ins*, como é o caso da *Rest Api*, que é um dos pontos principais para o funcionamento do aplicativo, sem contar a possibilidade de customização do total do código do sistema, fato que torna possível mudar tudo para adaptar às necessidades do aplicativo; para tal fato utilizamos o que conhecemos no *Wordpress* como categorização de *posts*.

O *Wordpress* por padrão possui modelos de postagens onde adicionamos títulos, texto e imagens manualmente. Com a possibilidade de categorizar as postagens, pode-se fazer com que o usuário do sistema selecione um tipo de categoria de postagem, por exemplo: Criamos uma categoria mãe chamada Alertas, onde serão mostrados todos os alertas do sistema e duas subcategorias, uma nomeada de “Assalto comigo”, e outra nomeada como “Assalto com vizinho”, onde caso o usuário selecione uma das duas no aplicativo a postagem será referente à categoria. Isto exemplifica o modo como é feito para os demais alertas.

O Modelo de usuário segue os padrões do *Wordpress*. Temos o usuário administrador com acesso total ao sistema, para adicionar ou excluir novos usuários, e o usuário padrão, ou seja, o Autor, que pode postar alerta do sistema ou comentar em alertas já postados. Como o *Wordpress* é um sistema web, ainda assim podemos permitir o acesso a esse sistema através de um navegador *Web*, agregando valor ao mesmo; e, para não permitir que qualquer pessoa não cadastrada acesse o sistema, configuramos a opção de que apenas usuários cadastrados no sistema podem acessar as páginas de alerta.

6. CONSIDERAÇÕES FINAIS

O Desenvolvimento de aplicativos híbridos para segurança compartilhada é de fato possível, visto que este pode sanar todas as necessidades dissertadas e até mesmo outras, além destas requeridas para o seu desenvolvimento. Verificamos a existência de diversas funcionalidades e, sem dúvida, esquecemos algumas, mas essa é a parte talvez mais importante de um desenvolvimento: modelar o seu protótipo para encontrar os erros. A princípio o modelo do aplicativo funciona apenas para um único grupo, em um único sistema *Wordpress*. Este fato se deve à questão de estar-se testando a possibilidade de funcionamento do mesmo. Comprovado este fato, assim estenderíamos este aplicativo para atender uma quantidade maior de usuários. O que mais nos motivou neste trabalho foi verificar que há uma grande quantidade de informações sobre o assunto e uma facilidade em codificar o aplicativo, tanto quanto utilizar a *Rest API*. Apesar de não termos programado o aplicativo por completo, fizemos diversos testes com o código e obtivemos sucesso, além de uma grande aceitação pelo público alvo, refletida na pesquisa sobre o *App*. Sem dúvida podemos concluir que este modelo de desenvolvimento de aplicações é ágil e eficaz, atingindo a maioria dos sistemas operacionais de *smartphones* e utilizando linguagens conhecidas por programadores *Web*, assim facilitando sua implementação e manutenção.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] *Brian Fling, Mobile Design and Development, First Edition, 2009*

[2] Raggett, Dave, Jenny Lam, Ian Alexander, and Michael Kmiec. *"Ragget on HTML4"*, "W3C." *World Wide Web Consortium (W3C)*. Addison Wesley Longman, 14 May 2014. Web. 28 Nov. 2016.

[3] SMITH, Steve; PASIC, Andy; ANDERSON, Rick. **Overview of ASP.NET Core MVC.** Disponível em: <https://docs.microsoft.com/en-us/aspnet/core/mvc/overview>. Acesso em: 29 nov. 2016.

[4] Williamson, Ken*, Learning AngularJS, First Edition* , 2015, O'Reilly Media, Inc

[5] W3C Standars for HTML JAVASCRIPT AND CSS. Disponível em: <https://www.w3.org> . Acesso em 16/12/2015.

[6] Wordpress Documentation Disponível em: <https://codex.wordpress.org> . Acesso em 29/11/2016.

[7] JSON Documentation Disponível em: <http://www.json.org/> Acesso em 29/11/2016.

[8] IONIC. **How to get the most out of Ionic.** Disponível em: <http://ionicframework.com/docs/guide/preface.html>. Acesso em: 16/12/2016.

[9] AngularJS Documentation and Guide, Disponível em: <https://docs.angularjs.org/guide/> Acesso em 11/12/2016.

[10] EWER, Tom. **The REST API:** And How It Could Change The World Forever. Disponível em: <wpmudev.org>. Acesso em 16/12/2016

[11] GOK, Nizamettin; KHANNA, Nittin. **Building Hybrid Android Apps with Java and JavaScript.** Sebastopol, Ca: O'reilly, 2013.

[12] WILLIAMSON, Ken. **Learning AngularJS:** A Guide To AngularJS Development. Sebastopol, Ca: O'reilly, 2015.