UNIVALI - UNIVERSIDADE DO VALE DO ITAJAÍ
CAMPUS KOBRASOL
CIÊNCIA DE COMPUTAÇÃO - BACHARELADO

Um Sistema de Gestão de Identidades Federadas e Centrado no Usuário alinhado ao Programa de Governo Eletrônico Brasileiro

Bolsista: André Luiz de Oliveira
Orientadora: Michelle Silva Wangham

Relatório Final

São José – 2016

SUMÁRIO

RESUMO

Um programa de Governo Eletrônico tem como princípio democratizar o acesso à informação, ampliar discussões e dinamizar a prestação de serviços públicos. Um ponto chave para os sistemas ou aplicações de governo eletrônico é a criação de um sistema de identificação, de autenticação e de autorização de usuários. Esses sistemas são conhecidos como sistemas de gestão de identidades, do inglês, *Identity Management (IdM) systems*. No Brasil, o programa GOV.BR, até o momento, não definiu qual será a estratégia nacional de gestão de identidades. Este projeto tem como objetivo prover a gestão de identidades adequada ao programa GOV.BR, por meio do desenvolvimento de um sistema de gestão identidades federadas e centrado no usuário, baseado no padrão SAML. O projeto envolverá (1) a análise das estratégias nacionais de gestão de identidades federadas adotadas em outros países visando identificar as soluções tecnológicas amplamente aceitas nestes países, (2) a modelagem do sistema de gestão de identidades federadas e centrado no usuário, (3) a implementação de um protótipo do sistema e a sua integração em um estudo de caso (aplicação de eGov) para verificação da sua aplicabilidade e (4) a realização de testes de software, que contribuirão para aferir a conformidade com o padrão SAML e os impactos da solução na usabilidade e funcionalidades da aplicação. Por fim, (5) a divulgação dos resultados desta pesquisa, na forma de artigos e relatórios técnicos.

Palavras-chave: Gestão de Identidades, Governo Eletrônico, Gestão de Identidades Federadas.

1. INTRODUÇÃO

Inúmeros países estão adotando políticas relacionadas à abertura e à transparência de seus governos. Segundo Thibeau e Reed (2009), um governo aberto é a doutrina política que defende que os negócios da administração pública devem ser abertos a todos os níveis, ao escrutínio público. O objetivo real de um governo aberto (*Open Government*) é o aumento da participação do cidadão e o envolvimento destes no governo (THIBEAU e REED, 2009).

O desenvolvimento de programas de Governo Eletrônico tem como princípio a utilização das modernas tecnologias de informação e comunicação (TICs) para democratizar o acesso à informação, ampliar discussões e dinamizar a prestação de serviços públicos com foco na eficiência e efetividade das funções governamentais. Os programas de Governo Eletrônico (e-Gov) bem sucedidos dependem do uso adequado das TICs empregadas nas organizações para promover os trabalhos colaborativos em prol de objetivos comuns (DAWES e PARDO, 2008).

Segundo Thibeau e Reed (2009), o crescente aumento das redes sociais, blogs, mensagens e tecnologias conhecidas como Web 2.0 têm o potencial de aumentar o fluxo de informações entre governos e cidadãos em ambos os sentidos, criando assim um cenário de colaboração.

Muitos países, tais como: Estados Unidos, Nova Zelândia, Itália, Reino Unido e Dinamarca, estão expandindo os seus programas de e-Gov aprimorando as suas infraestruturas de colaboração, sendo que o grande desafio está em garantir a interoperabilidade entre estas infraestruturas devido a heterogeneidade dos procedimentos e dos dados existentes entre a administração pública central e as locais (BALDONI, 2010). Esta diversidade pode tornar difícil a implantação destes programas (OECD, 2011).

No relatório das Nações Unidas sobre governos eletrônicos (UNITED NATIONS, 2014), o Brasil ocupa a 57º posição do ranking mundial de desenvolvimento de aplicações de eGov. Este relatório apresenta ainda um ranking que trata da utilização das aplicações por parte dos cidadãos, chamado E-Participação. Neste ranking, o Brasil ocupa a 24º posição (UNITED NATIONS, 2014).

Um ponto chave para os sistemas ou aplicações de governo eletrônico é a criação de um sistema de identificação, de autenticação e de autorização de usuários. Esses

sistemas são conhecidos como sistemas de gestão de identidades (*Identity Management (IdM) systems*) (BALDONI, 2010). A gestão de identidades pode ser entendida como o conjunto de processos e tecnologias usados para garantir a identidade de uma entidade ou de um veículo, garantir a qualidade das informações de uma identidade (identificadores, credenciais e atributos) e para prover procedimentos de autenticação, autorização, contabilização e auditoria (ITU-T, 2009).

Um dos problemas, no caso das aplicações e-Gov, é possibilitar que as aplicações (provedores de serviços) suportem a autenticação única (*Single Sign On*) de usuários. Comumente, as instituições do governo acabam duplicando o cadastro de pessoas já registradas. Outro problema que deve ser tratado no gerenciamento de identidades é a privacidade das informações (HANSEN et al. 2008). Em um cenário ideal, os usuários devem exercer o direito de determinar como suas informações serão manipuladas, informando quais informações poderão ser compartilhadas com terceiros, como esse compartilhamento deve ser feito e também indicando o período de tempo no qual essas informações poderão ficar disponíveis nos sistemas.

Sistemas de gestão de identidades federadas permitem o compartilhamento dos atributos do usuário e a autenticação única através de múltiplos domínios, tornando-se facilitadores para os sistemas governamentais [BALDONI 2012]. Nos últimos anos, alguns governos aprovaram estratégias nacionais de gestão de identidades baseadas no modelo federado buscando melhorar seus serviços de governo eletrônico, dentre estes, destacam-se: Nova Zelândia, Austrália, Canadá e Estados Unidos [OECD 2011].

Neste contexto, este projeto visa contribuir na área de gestão de identidades federadas, mas especificamente para aplicações do Programa de Governo Eletrônico Brasileiro.

2. OBJETIVOS

2.1 Objetivo Geral

Prover a gestão de identidades federadas adequada ao programa de governo eletrônico brasileiro, por meio do desenvolvimento de um sistema de gestão de identidades federadas centrado no usuário.

2.2 Objetivos específicos

De forma a alcançar o objetivo geral, os seguintes objetivos específicos foram definidos:

- Analisar as estratégias nacionais de gestão de identidades federadas de outros países visando identificar quais soluções tecnológicas atendem as necessidades de um modelo centrado no usuário e estão alinhadas ao programa eGov.BR;
- Conceber um sistema de gestão de identidades federadas centrado no usuário para as redes colaborativas governamentais brasileiras que atenda os requisitos impostos pelo e-PING;
- Verificar a aplicabilidade do sistema proposto por meio da implementação de um protótipo e da sua integração a um estudo de caso – aplicação web de eGov;
- Verificar os impactos do uso do protótipo na usabilidade e funcionalidades da aplicação e Gov.

3. FUNDAMENTAÇÃO TEÓRICA

3.1 Sistemas de Gestão de Identidade

Segundo Clauß e Köhntopp (2001), a identidade de uma pessoa é composta por uma grande quantidade de informações pessoais que caracteriza essa pessoa em diferentes contextos dos quais esta faz parte. A identidade é composta pela combinação de subconjuntos, chamados de identidades parciais, sendo que alguns identificam unicamente uma pessoa (p.ex. cpf) e outros não (p.ex. sexo). No contexto de um órgão do governo, a identidade pode estar associada com funções, privilégios, direitos e responsabilidades. Cabe salientar que uma mesma informação pessoal pode estar presente em diferentes identidades parciais.

Um sistema de gestão de identidades provê ferramentas para o gerenciamento dessas identidades parciais em um mundo digital. Segundo Chadwick (2009), a gestão de identidades consiste em um conjunto de funções e habilidades, como administração, descoberta e troca de informações, usadas para garantir a identidade de uma entidade e as informações contidas nessa identidade, permitindo assim que relações comerciais possam ocorrer de forma segura. Assim, um sistema de gerenciamento de identidades

consiste na integração de políticas e processos de negócios, resultando em um sistema de autenticação de usuários aliado a um sistema de gestão de atributos.

De acordo com Bhargav-Spantzel et al. (2007), o sistema de gerenciamento de identidades é caracterizado pelos seguintes elementos:

- Usuário: aquele que deseja acessar algum serviço;
- Identidade: conjunto de atributos de um usuário, que pode ser seu nome, endereço, filiação, data de nascimento, etc;
- Provedor de Identidades (*Identity Provider* – IdP): responsável por fazer a gestão da identidade de um usuário. Após o usuário passar por um processo de autenticação, este recebe uma credencial, dita identidade, que é reconhecida como válida pelos provedores de serviço;
- Provedor de Serviços (*Service Provider* – SP) oferece recursos a usuários autorizados, após verificar a autenticidade de sua identidade e após comprovar que a mesma carrega todos os atributos necessários para o acesso.

Segundo Jøsang e Pope (2005), os sistemas de gestão de identidades seguem modelos classificados como tradicional, centralizado, federado e centrado no usuário. O modelo tradicional ainda é amplamente utilizado nos sistemas atuais. Neste modelo, a autenticação é tratada de forma isolada por cada provedor de serviços. O usuário deve se autenticar em cada serviço que deseja utilizar, pois não existe compartilhamento de identidades entre os provedores de serviços. O modelo tradicional é amplamente utilizado nos atuais sistemas presentes na Internet. Neste modelo, a identificação do usuário é tratada de forma isolada por cada provedor de serviços (SP), o qual também atua como provedor de identidades (IdP). Cabe ao usuário criar uma identidade digital para cada SP que deseje interagir, não havendo assim o compartilhamento das identidades desses usuários entre diferentes SPs.

Apesar de amplamente adotado, o modelo tradicional (isolado) é custoso tanto para usuários quanto para SPs. Cada SP pode exigir um conjunto próprio de atributos para compor a identidade digital do usuário. Por outro lado, um conjunto comum de atributos pode ser exigido por diversos SPs, como nome da conta, senha, endereço, data de nascimento. Para os usuários, gerenciar inúmeras identidades é algo custoso. Primeiro, por ter que fornecer as mesmas informações diversas vezes, segundo, por ter que se preocupar em criar um nome de usuário e senha diferente para cada SP, uma vez

que usar a mesma senha por diversos provedores não é aconselhado (WANGHAM *et al.*, 2010).

O modelo centralizado surgiu como uma solução para a inflexibilidade do modelo tradicional e está fundamentado no compartilhamento das identidades dos usuários entre SPs e no conceito de autenticação única (Single Sign-on - SSO) (BHARGAV-SPANTZEL et al, 2007). Neste modelo, só existe um único IdP o qual é responsável por autenticar os usuários, sendo que todos os provedores de serviços devem confiar plenamente nas informações fornecidas por este IdP.

Visando contornar as dificuldades apresentadas pelo modelo centralizado, o modelo de identidades federadas está fundamentado sobre a distribuição da tarefa de autenticação dos usuários por múltiplos IdPs, estando estes dispostos em diferentes domínios administrativos. Um domínio administrativo pode representar um órgão do governo e é composto por usuários, SPs e um único IdP (CAMENISCH E PFITZMANN 2007).

A gestão de identidades federadas é uma abordagem para otimizar a troca de informações relacionadas a identidade através de relações de confiança construídas nas federações. Acordos estabelecidos entre IdPs garantem que identidades emitidas em um domínio sejam reconhecidas por SPs de outros domínios e o conceito de autenticação única é garantido mesmo diante de diferentes domínios. Dessa forma, o modelo de identidades federadas consegue oferecer facilidades para os usuários, pois evita que estes tenham que lidar com diversas identidades e passar diversas vezes pelo processo de autenticação. Para os SPs, o benefício é que estes poderão lidar com um número menor de usuários temporários (CAMENISCH E PFITZMANN 2007).

Jøsang e Pope (2005) consideram a gestão de identidades federadas como um modelo centrado nos Ids. Apesar de haver a distribuição das identidades por diversos provedores, informações desses usuários, uma vez liberadas para esses provedores, podem ser disponibilizadas a terceiros. O modelo centrado no usuário objetiva dar ao usuário o total controle sobre suas identidades digitais, contudo as principais propostas e implementações deste modelo fazem uso de um dos modelos apresentados anteriormente, sendo o modelo de identidades federadas o mais usado.

3.2 Comparativo de Estratégias Nacionais de Gestão de Identidades para Governo Eletrônico

A ONU com seu departamento de assuntos econômicos aplicam uma pesquisa nos 193 Estados membros, aborda assunto sobre o desenvolvimento do e-GOV (Governo Eletrônico). Após a pesquisa ser aplicada e seus resultados serem obtidos, é feito uma análise dos resultados e identificado pontos frágeis no desenvolvimento dos países, desafios a serem estabelecidos para os anos seguintes e elaboração de programas nacionais para aconselhar as políticas nacionais e práticas na área de inovação tecnológica. O resultado simplificado da pesquisa é um ranking de países com seus respectivos índices de desenvolvimento de e-GOV, assim conhecido como EGDI. Na tabela abaixo os resultado do ano de 2014:

Tabela 1. Características dos Países Analisados

Rank (2012)	Rank (2014)	País	Região	EGDI (2014)	Rank e-Particip.	Serviços online
1	1	Coréia do Sul	Asia	0,9462	2	3
12	2	Austrália	Oceania	0,9103	7	7
10	3	Singapura	Asia	0,9076	10	2
6	4	França	Europa	0,8938	4	1
2	5	Holanda	Europa	0,8897	1	8
18	6	Japão	Asia	0,8874	5	4
5	7	EUA	Américas	0,8748	9	5
3	8	Reino Unido	Europa	0,8695	6	10
13	9	Nova Zelândia	Oceania	0,8644	20	14
9	10	Finlândia	Europa	0,8449	25	18

Sistema de gestão de identidade (IdM) tem por intuito principal ser a fechadura de serviços, tecnologias, políticas e processos de negócios. Assim explicando melhor o que seria uma fechadura, é o mecanismo de autenticação na qual os usuários terão que ter suas chaves para terem acesso e requisitar o terem interesse. Toda porta tem sua fechadura, as portas são representam os ambientes que comportam a parte interessante que instiga o usuário à fazer solicitações. Para isso existem modelos diferentes de distribuição para IdM: tradicional ou isolado, centralizado, federado e centrado no usuário.

Mas, antes de comentar sobre os tipos de modelos é preciso entender quais os atores presentes nos modelos. Um deles é o IdP que segue a tarefa de autenticar o usuário, o SP prove serviços.

No modelo tradicional, o IdP e o SP, são mantidos no mesmo servidor, mesma estrutura física ou lógica na arquitetura, por isso é chamado de “isolado”. Enquanto no

modelo centralizado, o IdP fica em um único servidor dentro do domínio administrativo, assim sendo utilizado por um ou mais SPs contidos naquele domínio.

O IdM é um desejo de alguns paises e encontra-se em estágios diferentes de pais para pais. Japão já executa suas estratégia a apresenta desenvolvimento e implanta sua estratégia , mas ainda está nas fases iniciais. Já a Coreia do Sul possui uma classificação com cem porcento desenvolvida. Quanto a Austrália e Holanda apresentando resultados significativos do seu desenvolvimento e em implantação se situação em uma situação já concreta e pronta para finalizações de seu desenvolvimento.

Sobre os modelos IdM implantados nos paises, o único que aderiu ao modelo centrado no usuário foi a Nova Zelândia. A Nova Zelândia também possui o modelo federado, juntamente com ela, Austrália, EUA e Reino Unido também usam o modelo federado. O modelo Cetrelizado é utilizado pela maioria sendo: Coréia do Sul, Singapura, França, Holanda, Japão e Finlândia.

O IdM com relação ao Brasil, a ONU classificou Brasil como 54 no ranking e não definiu a estratégia nacional de gestão de identidades para e-Gov.

3.2 Aplicativos E-GOV

Em pesquisa realiza, foram encontrados aplicativos E-GOV que já estão sendo utilizados pelos brasileiros para facilitar, agilizar e democratizar processos do governo. Os aplicativos que estão em questão são: IRPF, Anatel consumidor, SRO Mobile dos Correios e Procon da cidade de Porto Alegre – RS.

Aplicativo da Receita Federal do Brasil foi criado para declarantes do Imposto de Renda Pessoa Física. A versão atual está disponível as funcionalidades: fazer declaração IRPF do ano vigente, rascunho do ano vigente, acompanhar declaração, esclarecimento de dúvidas com questionários respondidos e orientações.

Anatel Consumidor serve para consumidores cadastrarem e acompanhar suas reclamações contra prestadores de telecomunicações. Também permite o cadastro de pedidos de informações e sugestões sobre a Agência e acesso a questionários respondidos sobre os direitos do consumidor de serviços de telecomunicação.

Aplicativo dos Correios, feito para disponibilizar aos usuários dos serviços dos Correios. Consiste em disponibilizar de maneira automática as encomendas postadas na ECT - Empresa Brasileira de Correios e Telégrafos. O aplicativo também disponibiliza os serviços: identificação da encomenda com um nome que classifique melhor para o

usuário, rastro da encomenda no território brasileiro, notificações quando é alterado status da entrega, entre outras funcionalidades.

Aplicativo criado pelo Procon de Porto Alegre - RS para facilitar a captação de denúncias provenientes de consumidores em geral. Possibilita que o registro da denúncia seja feita em três passos para que o usuário tenha facilidade e rapidez no seu cadastro. O aplicativo também disponibiliza um rankings das empresas mais reclamadas de Porto Alegre, localização das unidades do Procon na cidade, dicas e dúvidas frequentes e uma área para o usuário entrar em contato com o Procon.

Alguns dos aplicativos citados nesta seção, como o Procon e Anatel, possuem semelhança ao aplicativo desenvolvido nesta pesquisa por tratarem de acompanhamento de solicitações.

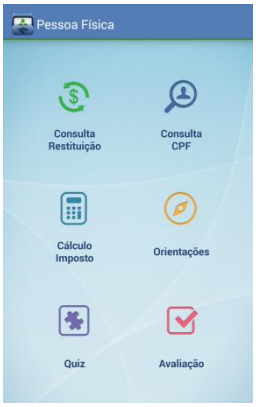


Figura 1 - IRPF

Figura 2 – Anatel

Figura 3 – Correios

Figura 4 - Procon

4. MATERIAIS E MÉTODOS DE PESQUISA

Nesta pesquisa, o método de experimentação foi empregado através do desenvolvimento e avaliação de um protótipo que envolve um mecanismo autenticação para aplicações E-GOV baseados em Serviços Web. A abordagem desta pesquisa foi qualitativa, pois não fez o uso de métodos e técnicas estatísticas.

Para a construção do protótipo foram utilizados os seguintes softwares de código aberto: o ambiente de desenvolvimento Eclipse, a ferramenta de modelagem Enterprise Achitect, os *frameworks* Spring e AngularJS para desenvolvimento de Serviços Web, Aplicação Web e Interface Web, respectivamente, e os servidores de aplicação Tomcat e Http Server. A gestão desse projeto se deu através de reuniões periódicas entre orientador e bolsista. A ferramenta de apoio à gestão de projetos Trello foi utilizada para acompanhamento de todas as etapas especificadas a seguir.

4.1 Estudo bibliográfico

Esta primeira etapa foi destinada à formação do acadêmico-pesquisador, através do estudo das especificações relacionadas à arquitetura orientada a serviços rest, à arquitetura e desenvolvimento de aplicativos mobile e à autenticação centrada no usuário e da literatura que trata de gerenciamento de identidades e mecanismos de autenticação única (SSO). Textos de apoio foram indicados pela orientadora e reuniões periódicas foram realizadas para auxiliar a consolidação da base teórica da pesquisa. Como indicador físico desta etapa, tem-se o texto da Seção 3. Nesta etapa, foi realizado ainda um estudo dos *frameworks* para desenvolvimento de aplicativos móbile de código aberto, para o desenvolvimento do mecanismo de autenticação federado e para o desenvolvimento do serviço rest, respectivamente, os frameworks IONIC, MITREid Connect e Spring foram os selecionados.

4.2 Implementação do Protótipo

Esta etapa de implementação foi divida em quatro atividades conforme descritas a seguir:

4.2.1 - Implementação da aplicação de registro de ocorrências

O aplicativo é baseado à serviço. Sendo assim, a aplicação *front-end* (parte visual da aplicação: páginas, componentes visuais, estilo de cores e botões) armazenada no dispositivo do usuário e as informações do aplicativo ficarão armazenadas em um servidor disponibilizado via serviço web. As duas partes, assim chamas de camadas, se comunicarão via *HTTP*, utilizando seus métodos(*GET*, *SET*, *PUT*, *DELETE*). Porém, temos uma exceção neste arquitetura de serviço por termos o mecanismo de autenticação contido também no cliente(dispositivo *smartphone*) com suas credenciais.

O Front-End desenvolvendo com as linguagens estáticas HTML e CSS para definir layouts e estilo das páginas. Para o controle das ações de tela utilizado a linguagem JavaScript. Para facilitar o desenvolvimento utilizadou-se framework IONIC desenvolvido em Javascript dedicado para criação de aplicativos para plataformas Android e IOS. O IONIC é uma evolução do Cordova, desenvolvido em java e JavaScript, a vantagem de utilizá-lo é criação de plugins e integrar às plataformas, no caso da plataforma Android desenvolvido em Java e IOS desenvolvido em Objective-C. Para solução da autenticação, foi feito experimento com desenvolvendo do plugin que conterá a implementação da autenticação. O IONIC é integrado com o framework AngularJS e segue suas arquitetura no projeto inicial. O AngularJS é implementado em

Javascript e facilita na controle das páginas com eventos, listagens, encapsulamento de objetos seguindo padrões do formato JSON, separação de camadas, acesso à camada HTTP, entre muitas facilidades disponíveis.

A back-end desenvolvido e disponibilizado via serviços REST. Para isso, foi desenvolvido em java utilizando o framework Spring e executada em um servidor de aplicação, como um Tomcat. O Spring possui métodos implícitos que faz o REST ser implementado com mais facilidade e de claro entendimento na manutenção. O armazenamento feito em um base de dados, com a utilização do SGBD MySQL.

4.2.2 – Configuração de serviço de gestão de identidade Firebase Auth

Na plataforma Firebase da Google é possível criar um projeto e nele utilizar recursos da plataforma. Um dos recursos é a gestão de identidade dos usuários. Então, com a utilização do recurso Auth foi possível validar a gestão de identidade com o Google.

4.2.3 – Implementação de uma Aplicação Cliente de autenticação com Firebase Auth

Para fim de validação do serviço de autenticação do Firebase Auth, que é o serviço de gestão de identidade que o Google disponibiliza, foi desenvolvido uma aplicação web com o framework Angular JS que exige autenticação para acessar a área privada. Disponibiliza acesso via registro ou login com o a sessão da conta Google.

4.2.4 – Configuração da aplicação de gestão de identidade MITREid Connect

MITREid é uma implementação código aberto do OpenID Connect e OAuth 2.0 do MITRE Corporation e MIT Internet Confiança Consortium (ITC). Para representar o servidor de autenticação federado o OpenID Connect Server foi o escolhido. No seu repositório no GIT possui diversas implementações, a escolhida foi desenvolvida em linguagem Java e com o framework Spring. Esta aplicação foi configurada à IDE de desenvolvimento Eclipse, compilada e gerado seu build para ser posto em um servidor web Tomcat. Para validar o servidor de autenticação, também possui em seu repositório uma aplicação que simula a conexão de clientes. Esta foi configurada à IDE eclipse e executado no servidor web Tomcat.

4.3 Avaliação do Protótipo

Durante o desenvolvimento do protótipo, testes de unidade foram realizados. Os erros encontrados foram corrigidos e os testes foram refeitos. Alguns testes de integração foram realizados, porém, com o protótipo não foi 100% finalizado, alguns

casos de testes previstos não foram executados, dentre estes, os testes de integração com provedor de identidade.

4.4 Documentação e Divulgação de Resultados

Os relatórios parciais, alguns resumos dos estudos realizados e o presente relatório foram redigidos nesta etapa.

5. RESULTADOS E DISCUSSÕES

A Figura 5 ilustra a tela de autenticação do aplicativo mobile e a Figura 6 é a tela de autenticação com as autenticações federadas.

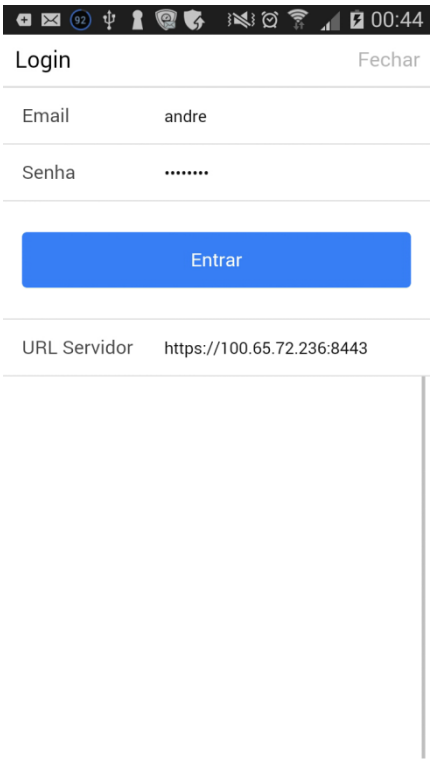


Figura 5 - App Login

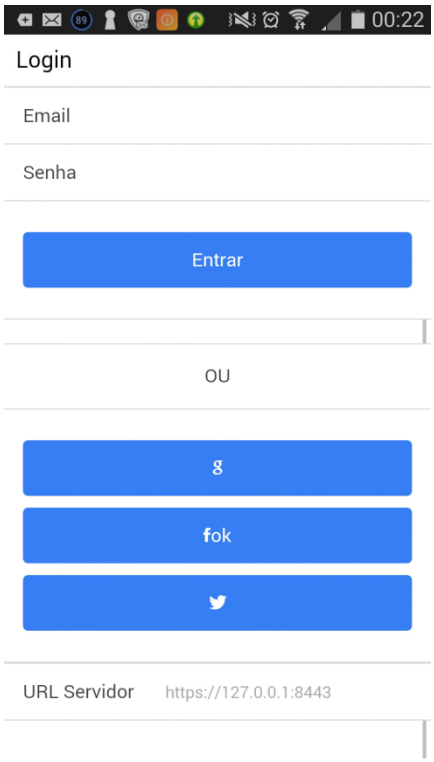


Figura 6 – Login com Aut. federada

Ao entrar no ambiente privado o usuário tem a possibilidade de entrar em ocorrências, lista de usuários ou sair, neste caso fazer logoff. Conforme mostrado na Figura 7.

Figura 7 – Menu App

As Figuras 8 e 9 ilustram a lista de usuários e ocorrências. Nestas telas usuário tem a possibilidade de adicionar novos, editar os existentes ou deletar o registro.

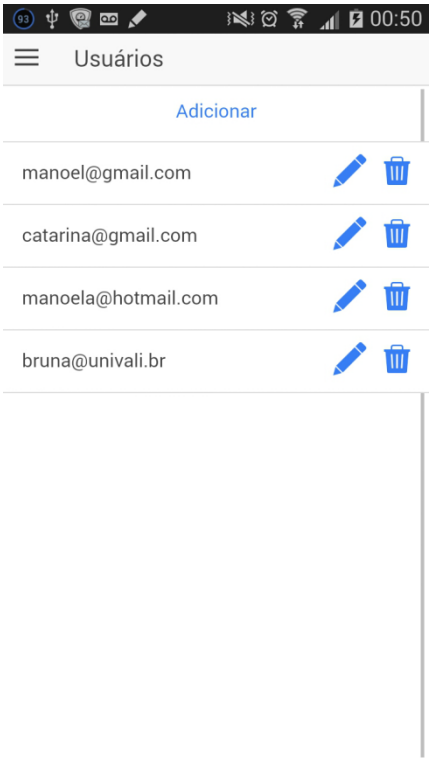


Figura 8- Lista de usuários

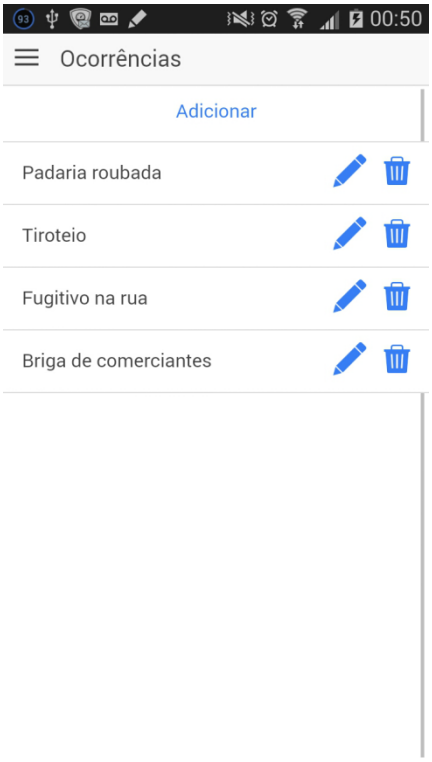


Figura 9 – Lista de ocorências

As Figuras 10 e 11 mostra as telas de edição e adição à lista de usuário e ocorrências.

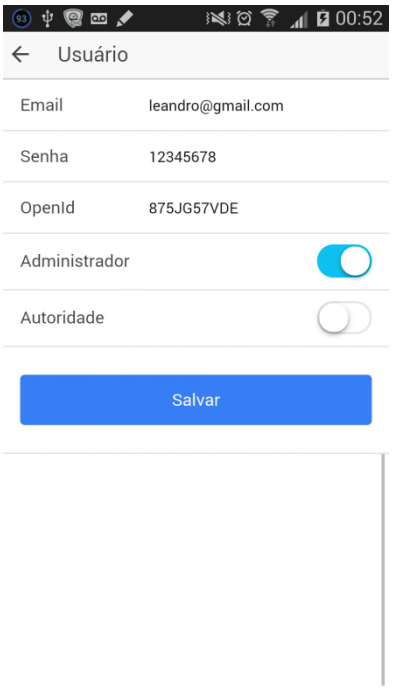


Figura 10 – Cadastro Usuário

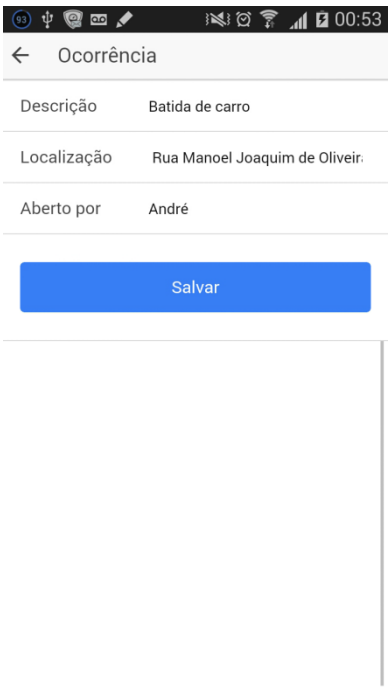


Figura 11 – Cadastro Ocorrência

Para demonstrar a execução e testar o serviço rest que o aplicativo busca as informações foi utilizado o Postman, plugin do Google Chrome. A Figura 12 mostra o resultado da requisição https://127.0.0.1:8443/occurrence/list.

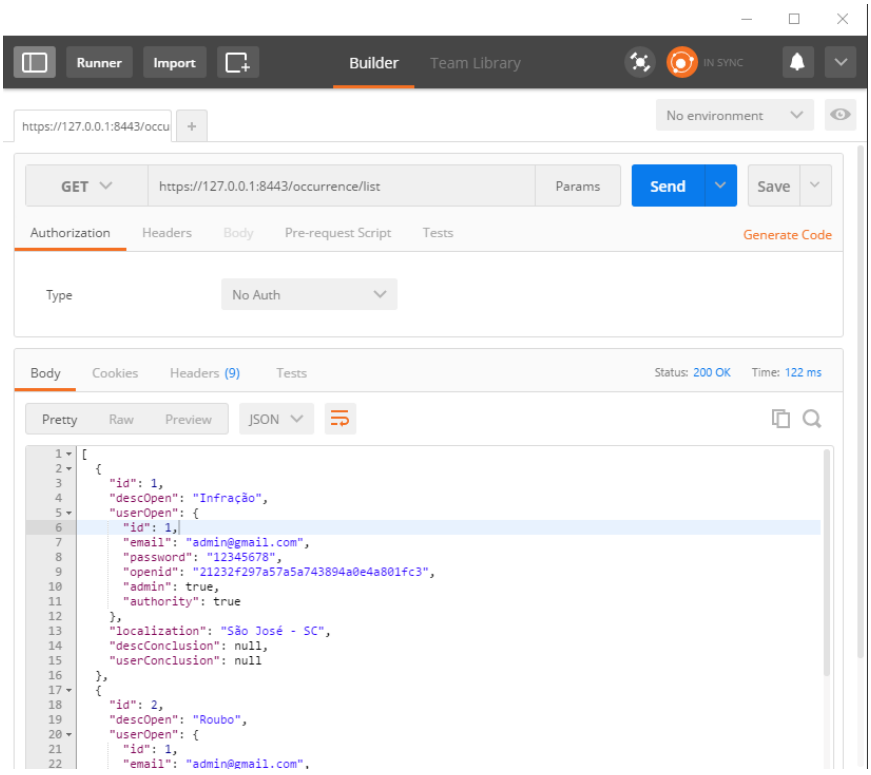


Figura 12 – Resultado Aplicação Servidor

O console do Firebase é mostrado na Figura 13. É possível perceber na imagem que é selecionado o recurso Auth e foi ativado para autenticação Google.

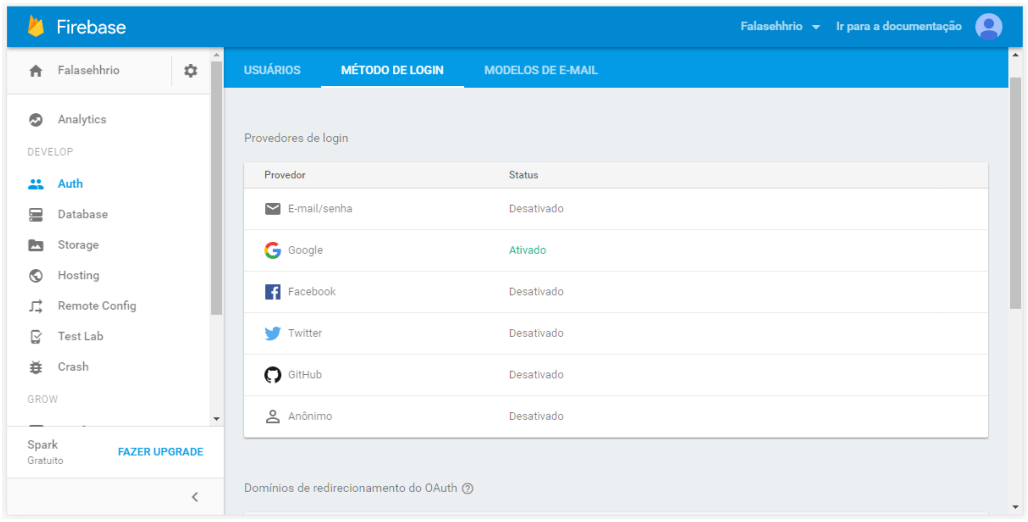


Figura 13 – Firebase Auth

Após ativação do Auth na plataforma Firebase foi possível efetuar autenticação no protótipo implementado, mostrado na Figura 14 e a Figura 15 mostra a página do Google na qual o usuário deve dar permissão de acesso aos seus dados.

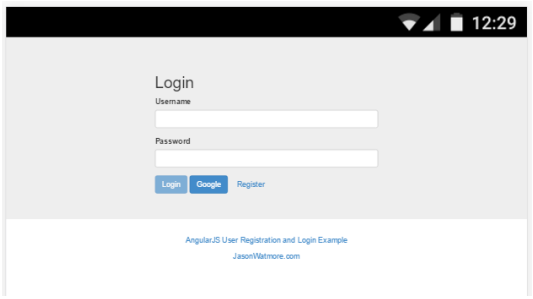


Figura 14 – Login Web Firebase

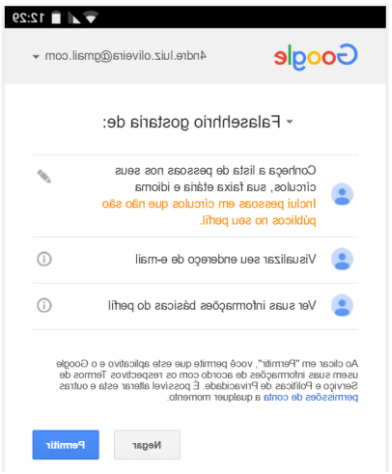


Figura 15 – Credencial Google

Após o experimento realizado com o Firebase deu-se inicio aos experimentos com a utilização do MITREid para criação do servidor federado de autenticação. A Figura 16 mostra a aplicação MITREid Server em execução e com o cliente Simple Web App já registrado.

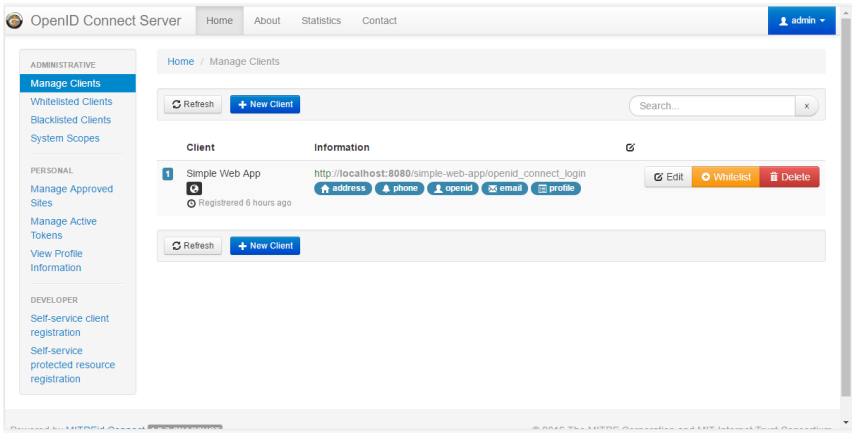


Figura 16 – MITREid Server

A Figura 17 mostra a aplicação cliente Simple Web App, qual se conecta ao Server para validar a autenticação federada.

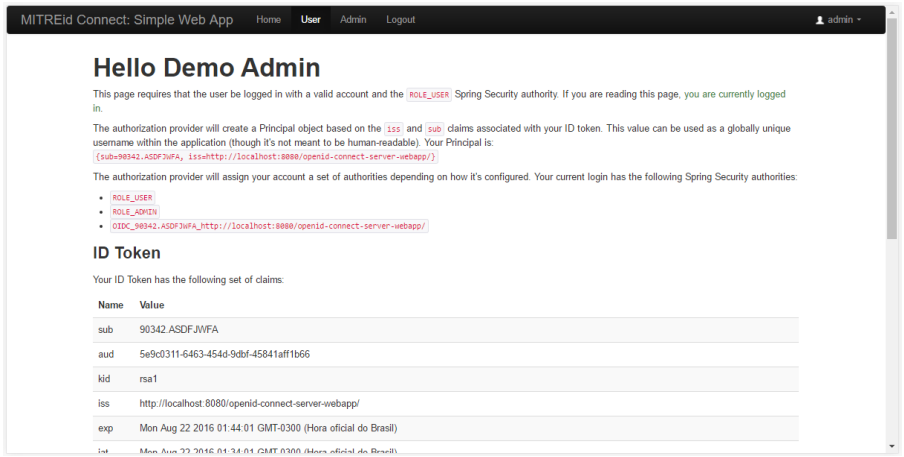


Figura 17 – Informações do usuário autenticado

Para o usuário se autenticar ao Server MITEid é necessário informar sua URL que está acessível. Como foi feito na Figura 18.

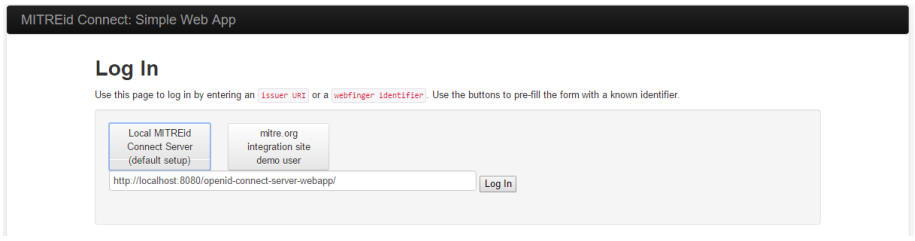


Figura 18 – URL do servidor de autenticação

Então, o usuário é redirecionado para o servidor de autenticação. Na página que é visualizada na Figura 18 o usuário deve selecionar as informações que deseja

disponibilizar para a aplicação cliente, semelhante ao mecanismo de autenticação do Firebase.

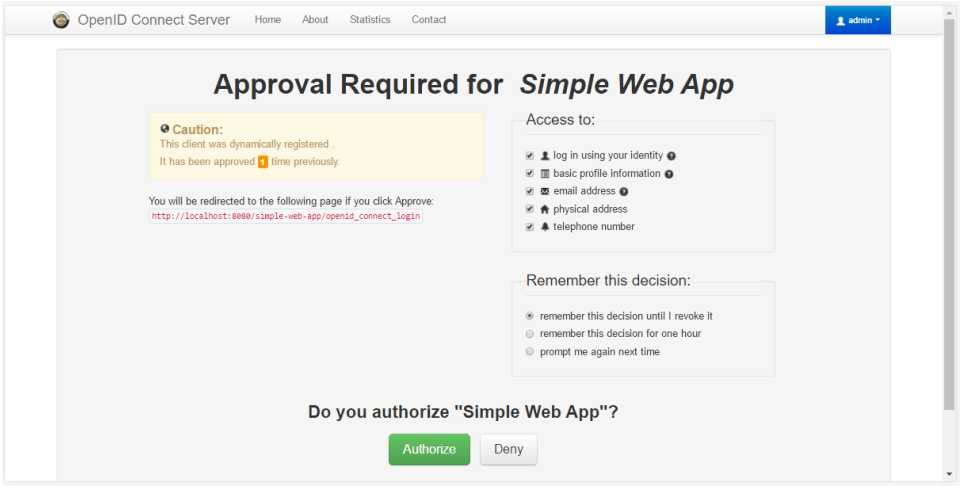


Figura 19 – Credencial MITREid Server

6. CONSIDERAÇÕES FINAIS

O uso do modelo centrado no usuário em programas de governo participativo é interessante devido a possibilidade de o usuário selecionar quais informações deseja liberar aos provedores de serviços, respeitando assim a privacidade dos usuários. Na solução proposta, por meio do modelo centrado no usuário, os cidadãos terão a possibilidade ainda de interagir com as aplicações e-Gov a partir de serviços de autenticação comumente utilizados, como por exemplo, o *framework* OpenID Connect ou o *Facebook Connect*.

Na solução proposta, o padrão SAML será usado para representar informações de segurança (credenciais de autenticação) na forma de asserções e na troca dinâmica de informações de segurança entre parceiros da federação governamental. Para que os processos de colaboração G2B, G2G e G2C se concretizem, o sistema proposto deve ainda contemplar um modelo de gestão de confiança (*Trust Framework*) que definirá como as relações de confiança entre os domínios administrativos deverão ser estabelecidas, como os níveis de garantia (*Level of Assurance - LoA*) podem ser definidos e avaliados e quais regras comuns são necessárias para operação da federação de serviços governamentais.

7. SUGESTÕES PARA DESDOBRAMENTO DA PESQUISA

Os experimentos de integração do aplicativo mobile com servidor de autenticação não foi concluída, por ocorrência de erros inesperados que necessitam de uma analise detalhada. Por este motivo foi implementado a aplicação web para validar a autenticação ao Server MITREid.

Para a aplicação mobile dar continuidade e implementação de melhorias, sugere-se integrar a câmera e a localização do dispositivo ao aplicativo desenvolvido. Isso facilitará o uso e gerará acertividade para as informações assim cadastradas.

Pesquisar sobre integração do MITREid à plataforma Firebase. Visto que, o Facebook, Google, Github e Twiter possuiem integração com a plataforma, verificar o possibilidade adicionar o servidor de gestão de identidide MITREid à Firebase Auth.

REFERÊNCIAS

BALDONI, R. Federated Identity Management Systems in e-Government: the Case of Italy. **Electronic Government**: An International Journal. v. 8, n. 1, 2010.

BHARGAV-SPANTZEL, A., CAMENISCH, J., GROSS, T., e SOMMER, D. User centricity: a taxonomy and open issues. **Journal of Computer Security**, v. 15, n. 5, p. 493–527, 2007.

CAMENISCH, J; PFITZMANN, B. Federated Identity Management. In: PETKOVIĆ, M; JONKER, W (eds.). **Security, Privacy, and Trust in Modern Data Management**. Berlin; Heidelberg: Springer Verlag, 2007. p. 213-238, cap. 5.

CHADWICK, D. Federated identity management. In: ALDINI, A.; BARTHE, G.; GORRIERI, R (eds). **Foundations of Security Analysis and Design V**. Berlin; Heidelberg: Springer-Verlag, 2009. p. 96–120.

CLAUß, S. e KÖHNTOPP, M. Identity management and its support of multilateral security. **Computer Networks**, v. 37, n. 2, p. 205–219, 2001.

DAWES, S. S. e PARDO, T. A. Advances in Digital Government Technology, Human Factors, and Policy, chapter Building Collaborative Digital Government Systems Systemic: constraints and effective practices, pages 259-273. Springer, US, 2008.

HANSEN, M.; SCHWARTZ, A.; COOPER, A. Privacy and identity management. **Security Privacy**, IEEE, v. 6, n. 2, p. 38 –45, 2008.

ITU. NGN identity management framework. [S.l.]: International Telecommunication Union (ITU), 2009. Recommendation Y.2720.

OECD. National Strategies and Policies for Digital Identity Management in OECD Countries. OECD Digital Economy Papers, No. 177, OECD Publishing, 2011.

THIBEAU, D. e REED, D. Open trust frameworks for open government: Enabling citizen involvement through open identity technologies. White paper, OpenID Foundation and Information Card Foundation. 2009.

WANGHAM, Michelle S. MELLO, Emerson Ribeiro de. BÖGER, Davi da Silva. GUERIOS, Marlon. FRAGA, Joni da Silva. Gerenciamento de Identidades Federadas. In: X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 10, 2010, Fortaleza. Anais do X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. Porto Alegre: Sociedade Brasileira e Computação, 2010. p. 447-460.

APÊNDICE – Descrição dos Projetos

Aplicação smartphone para registro de ocorrências públicas

A aplicação servirá para pessoas civis registrarem irregularidades que ocorrem no meio em que vivem. Irregularidades como carros estacionados em local proibido, assaltos em andamento, crimes contra o patrimônio público, entre outros crimes que são observados no dia-dia das pessoas e ficam sem ação de prosseguir a denúncia. O tratamento do registro da denúncia será efetuado por uma autoridade mais próxima. A autoridade será um policial, guarda municipal ou responsável de manter a order na região, que terá acesso especial ao aplicativo para visualizar e adotar as ocorrências, assim ser solucionadas e encaminhadas com o registro do BO (boletim de ocorrência). O registro da ocorrência será feito de forma simplificada e rápida para evitar prolongamento e talvez a perda de informação do que ocorreu. O usuário registrador irá acessar o aplicativo via *smartphone*, terá a opção de tirar até quatro fotos do ocorrido, descrever com poucas palavras a ocorrência, modificar a localização que foi capturada automaticamente pelo GPS do dispositivo e enviar para a central de registros. É previsto que este procedimento de registro da ocorrência tenha um tempo de duração de 5 min. Em comparação com o processo de denuncia atual serpa muito mais rápido e preciso, pois dará a localização exata do local e as autoridades conseguiram com o link fazerem a rota até o local.

Arquitetura da aplicação

O aplicativo será baseado à serviço. Sendo assim, a aplicação *front-end* (parte visual da aplicação: páginas, componentes visuais, estilo de cores e botões) ficará armazenada no dispositivo do usuário e as informações que populam o aplicativo ficarão armazenadas em um servidor disponibilizado na internet. As duas partes, assim chamas de camadas, se comunicarão via *HTTP*, utilizando seus métodos(*GET, SET, PUT, DELETE*). Porém, temos uma exceção neste arquitetura de serviço por termos o mecanismo de autenticação contido também no cliente(dispositivo *smartphone*) com suas credenciais.

O Front-End será desenvolvendo com as linguagens estáticas HTML e CSS para definir layouts e estilo das páginas. Para o controle das ações de tela será utilizado a linguagem Javascript. Para facilitar o desenvolvimento será utilizado framework IONIC desenvolvido em Javascript dedicado para criação de aplicativos para plataformas Android e IOS. O IONIC é uma evolução do Cordova, desenvolvido em java e javascript, a vantagem de utilizá-lo é podermos criar plugins e integrar às plataformas, no caso da plataforma Android desenvolvido em java e IOS desenvolvido em Objective-C. Para solução da autenticação, será desenvolvido o plugin que conterá a implementação da autenticação. O IONIC é integrado com o framework AngularJS e segue suas arquitetura no projeto inicial. O AngularJS é implementado em Javascript e facilita na controle das páginas com eventos, listagens, encapsulamento de objetos seguindo padrões do formato JSON, separação de camadas, acesso à camada HTTP, entre muitas facilidades disponíveis.

A back-end será desenvolvido e disponibilizado vias serviços REST. Para isso, será desenvolvido em java utilizando o frameword Spring.io e ficará sendo executada em um servidor de aplicação, como um Tomcat. O Spring possui métodos implícitos que faz o REST ser implementado com mais facilidade e de claro entendimento na manutenção. O armazenamento será feito em um base de dado, com a utilização do SGBD MySQL.

Requisitos funcionais:

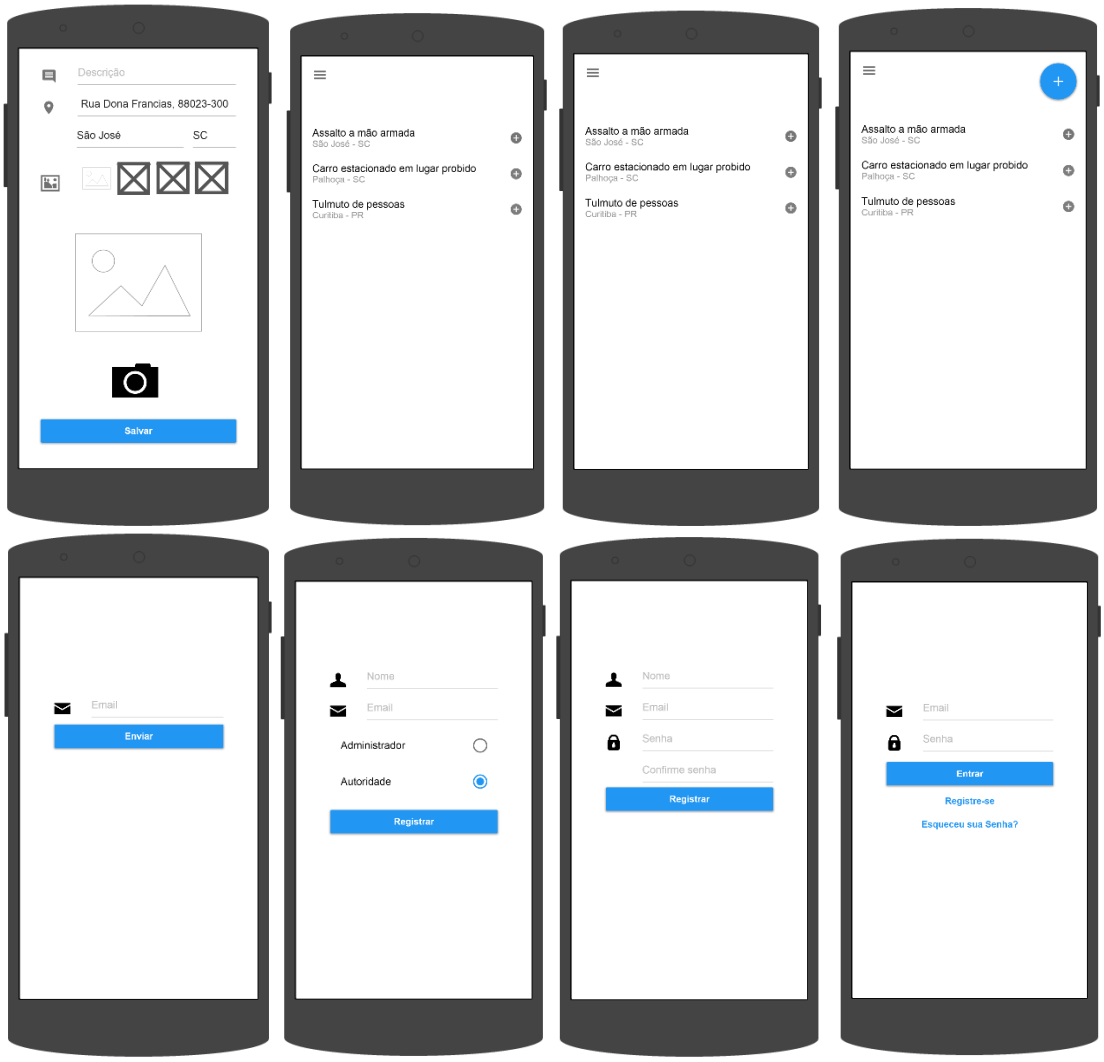
- RF001 - O sistemas deverá conter cadastro de usuários
    - RN001 - Informações do usuário: nome, email, senha
    - RN002 - Confirmação de senha para registro do usuário
    - RN003 - O sistemas conterá três tipos de perfis:
        - Administrador - Criado somente por usuários administrador no ambiente administrador.
        - Registrador - Criado somente no acesso público, primeira na página inicial do aplicativo.
        - Autoridade - Criado somente por administradores no ambiente administrador
- RF002 - O sistemas deverá conter tela principal de acesso ao sistema
- RF003 - O sistemas deverá conter recuperação da senha do usuário
- RF004 - O sistemas deverá conter página principal do ambiente registrador

- RF005 - O sistemas deverá conter página principal do ambiente autoridade
- RF006 - O sistemas deverá conter página principal do ambiente administrador
- RF007 - O sistemas deverá conter cadastro de ocorrências
    - RN004 - Informações da ocorrência: descrição, localização e fotos
- RF008 - O sistemas deverá conter visualização da ocorrências
- RF009 - O sistemas deverá filtrar as ocorrências
    - RN005 - Tipos de filtros: por período, por status
- RF010 - O sistemas deverá conter atribuição de ocorrências
- RF011 - O sistemas deverá conter resolução de ocorrências
    - RN006 - Informação da resolução: conclusão da ocorrência

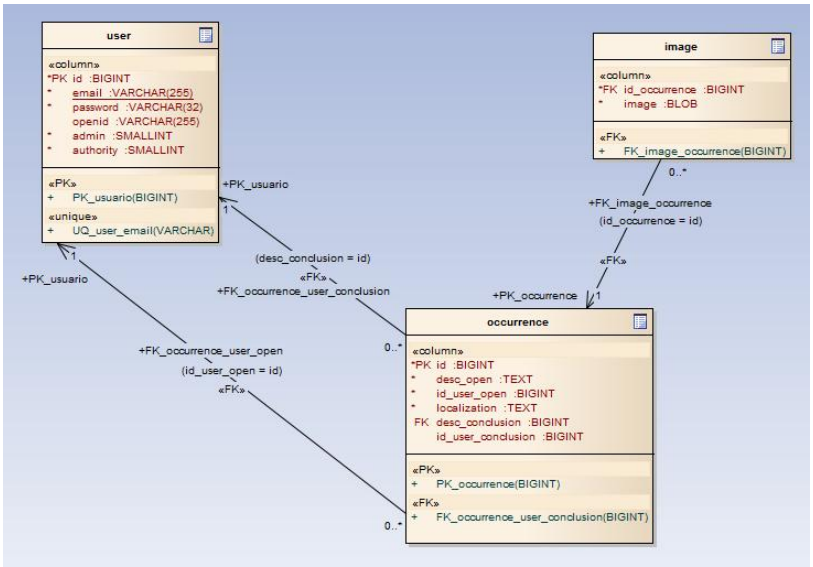
Requisitos não funcionais:

- RNF001 - Linguagem de programação Java para desenvolvimento da camada de serviço e implementação do plugin de autenticação
- RNF001 - Framework Spring para desenvolver métodos REST
- RNF001 - Maven para gerar o .war que será implamntado no servidor de aplicação
- RNF001 - Servidor de aplicação Tomcat para executar camada de serviço
- RNF001 - Linguagem de programação Javascript para desenvolvimento da camada de controle das páginas.
- RNF001 - Framework IONIC para desenvolvimento do aplicativo para plataforma Android
- RNF001 - AngularJS para fazer o controle dos dos evento e encapsulamento das informações da tela
- RNF001 - Plugman será utilizado para geração de plugins para plataforma Cordova (IONIC implementa)

Protótipos de tela:



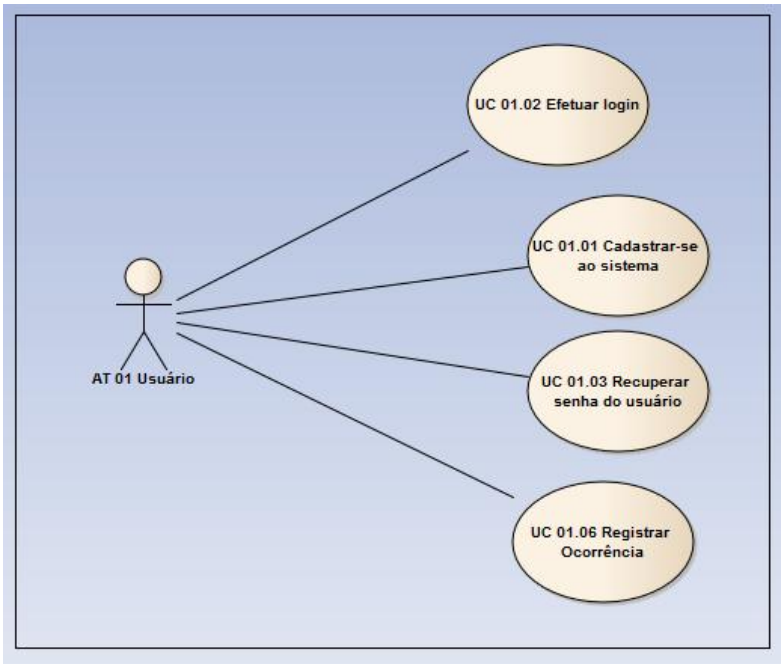
Modelagem ER:

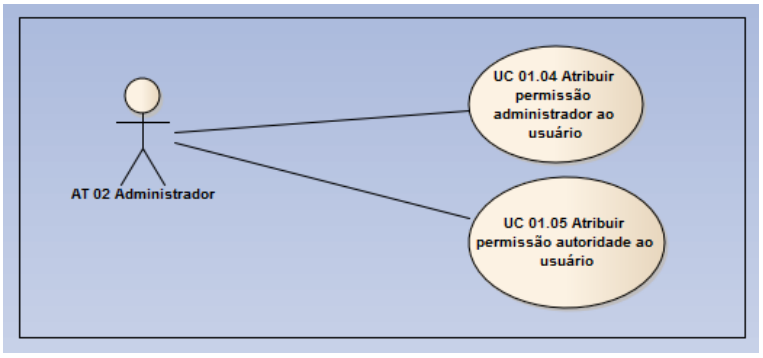


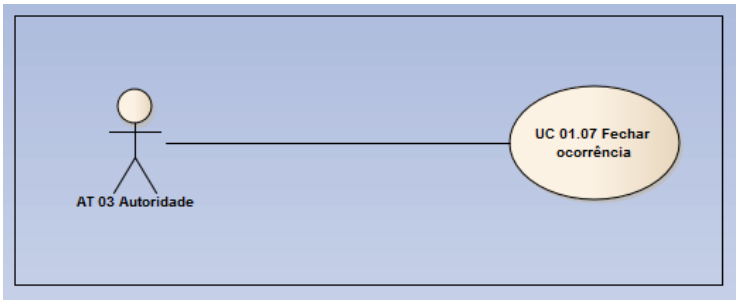
Script SQL da base de dados:

```
CREATE TABLE falasehhrio.image
(
	id_occurrence BIGINT NOT NULL,
	hash_image TEXT NOT NULL,
	KEY (id_occurrence)
);

CREATE TABLE falasehhrio.occurrence
(
	id BIGINT NOT NULL AUTO_INCREMENT,
	desc_open TEXT NOT NULL,
	id_user_open BIGINT NOT NULL,
	localization TEXT NOT NULL,
	desc_conclusion TEXT,
	id_user_conclusion BIGINT,
	PRIMARY KEY (id),
	KEY (desc_conclusion)
);

CREATE TABLE falasehhrio.user
(
	id BIGINT NOT NULL AUTO_INCREMENT,
	email VARCHAR(255) NOT NULL,
	password VARCHAR(32) NOT NULL,
	openid VARCHAR(255),
	admin SMALLINT NOT NULL,
	authority SMALLINT NOT NULL,
	PRIMARY KEY (id),
	UNIQUE UQ_user_email(email)
);

ALTER TABLE falasehhrio.image ADD CONSTRAINT FK_image_occurrence
	FOREIGN KEY (id_occurrence) REFERENCES falasehhrio.occurrence
(id)
;

ALTER TABLE falasehhrio.occurrence ADD CONSTRAINT
FK_occurrence_user_conclusion
	FOREIGN KEY (desc_conclusion) REFERENCES falasehhrio.user (id)
;
```

Casos de Uso:







Nome do Caso de Uso	UC 01.01 Cadastrar-se ao sistema
Breve Descrição	O usuário pode efetuar cadastro no provedor OpenID do aplicativo
Ator(es) Primário	Usuário
Pré-condições	Nenhuma
Fluxo principal:	1. O usuário clica em “Cadastrar-se”.

	2. O sistema apresenta o formulário de cadastro. 3. O usuário preenche o formulário e salva. 4. O sistema salva os dados na base de dados. 5. O sistema autentica o usuário.
Fluxos alternativos e exceções	Fluxo alternativo (3): Se o usuário preencher as informações sem preencher os campos obrigatórios. 1. O sistema apresenta mensagem de erro. 2. Volta para o passo 2.
Pós-condições	O sistema apresenta a tela inicial com o usuário autenticado.

Nome do Caso de Uso	UC 01.02 Efetuar login
Breve Descrição	Usuário deve efetuar login.
Ator(es) Primário	Usuário
Pré-condições	Nenhuma
Fluxo principal:	1. O usuário seleciona seu provedor 2. OpenID, exemplo, Gmail ou Yahoo. 3. O navegador do usuário é redirecionado para a página do provedor de identidades. 4. O usuário se autentica no provedor. 5. O sistema verifica a resposta do provedor. 6. O sistema cria a sessão para o usuário.
Fluxos alternativos e exceções	Fluxo alternativo (1): Caso o usuário não possua provedor OpenID 1. Executa o UC 01.01
Pós-condições	O sistema apresenta a tela inicial com o usuário autenticado.

Nome do Caso de Uso	UC 01.03 Recuperar senha do usuário
Breve Descrição	Usuário pode recuperar sua senha.
Ator(es) Primário	Usuário
Pré-condições	O usuário ter conta no provedor OpenID do Aplicativo.

Fluxo principal:	1. O usuário clica em “Esqueci minha senha”. 2. O sistema apresenta o formulário. 3. O usuário preenche o seu e-mail. 4. O sistema envia um link para o e-mail do usuário. 5. O usuário acessa o link. 6. O usuário digita sua nova senha e confirma.
Fluxos alternativos e exceções	Fluxo alternativo (3): Se o usuário preencher as informações de maneira incorreta. 1. O sistema apresenta mensagem de erro. 2. Volta para o passo 2.
Pós-condições	O sistema apresenta mensagem de sucesso.

Nome do Caso de Uso	UC 01.04 Atribuir permissão administrador ao usuário
Breve Descrição	O Administrador pode atribuir permissão administrador ao usuário.
Ator(es) Primário	Administrador
Pré-condições	1. Administrador estar autenticado. 2. Existir um usuário cadastrado. 3. Usuário a ser atribuído não ser administrador
Fluxo principal:	1. O administrador acessa seção usuários. 2. O sistema apresenta a lista de usuários. 3. O administrador seleciona o usuário. 4. O sistema apresenta informações do usuário. 5. O administrador seleciona opção “administrador”
Fluxos alternativos e exceções	Fluxo alternativo (3): Se o usuário selecionou o usuário incorreto. 1. Administrador seleciona “cancelar” 2. Volta para o passo 2.
Pós-condições	Usuário atribuído recebe permissões de “administrador”

Nome do Caso de Uso	UC 01.05 Atribuir permissão autoridade ao usuário
Breve Descrição	O Administrador pode atribuir permissão autoridade ao usuário.
Ator(es) Primário	Administrador
Pré-condições	1. Administrador estar autenticado 2. Existir um usuário cadastrado 3. Usuário a ser atribuído não ser autoridade
Fluxo principal:	1. O administrador acessa seção usuários 2. O sistema apresenta a lista de usuários. 3. O administrador seleciona o usuário 4. O sistema apresenta informações do usuário. 5. O administrador seleciona opção “autoridade”
Fluxos alternativos e exceções	Fluxo alternativo (3): Se o usuário selecionou o usuário incorreto. 1. Administrador seleciona “cancelar” 2. Volta para o passo 2.
Pós-condições	Usuário atribuído recebe permissões de “autoridade”

Nome do Caso de Uso	UC 01.06 Registrar Ocorrência
Breve Descrição	O Usuário pode registrar ocorrência.
Ator(es) Primário	Usuário
Pré-condições	1. Usuário estar autenticado
Fluxo principal:	1. Usuário clica em “registrar ocorrência” 2. Sistema apresenta formulário 3. O usuário preenche o formulário e salva. 4. O sistema salva os dados na base de dados.
Fluxos alternativos e exceções	Fluxo alternativo (3): Se o usuário preencher as informações sem preencher os campos obrigatórios. 1. O sistema apresenta mensagem de erro. 2. Volta para o passo 2.
Pós-condições	O sistema apresenta a tela inicial

Nome do Caso de Uso	UC 01.07 Fechar ocorrência
Breve Descrição	A Autoridade pode fechar ocorrência.
Ator(es) Primário	Autoridade
Pré-condições	1. Autoridade estar autenticada
Fluxo principal:	1. A autoridade seleciona a ocorrência 2. O sistema apresentar informações da ocorrência e formulário de conclusão. 3. A autoridade preenche o formulário e salva. 4. O sistema salva os dados na base de dados.
Fluxos alternativos e exceções	Fluxo alternativo (3): Se a autoridade preencher as informações sem preencher os campos obrigatórios. 1. O sistema apresenta mensagem de erro. 2. Volta para o passo 2.
Pós-condições	O sistema apresenta a tela inicial

Código-fonte da classe User:

```
public class User implements Model {
	private int id;
	private String email;
	private String password;
	private String openid;
	private boolean admin;
	private boolean authority;
	//Construtores
	// Gets e Sets
	@Override
	public String toString() {
		return "User {\"id\":\"" + id + "\", email\":\"" + email + "\",
password\":\"" + password + "\", openid\":\""
				+ openid + "\", admin\":\"" + admin + "\",
authority\":\"" + authority + "}";
	}
	@Override
	public List<Object> toArray() {
		List<Object> list = new ArrayList<>();
		list.add(this.email);
		list.add(this.password);
		list.add(this.openid);
		list.add(this.admin);
		list.add(this.authority);
		return list;
	}
```

```
	@Override
	public List<Object> toArrayWithoutId() {
		List<Object> list = new ArrayList<>();
		list.add(this.email);
		list.add(this.password);
		list.add(this.openid);
		list.add(this.admin);
		list.add(this.authority);
		return list;
	}
	@Override
	public List<Object> toArrayIdLast() {
		List<Object> list = new ArrayList<>();
		list.add(this.email);
		list.add(this.password);
		list.add(this.openid);
		list.add(this.admin);
		list.add(this.authority);
		list.add(this.id);
		return list;
	}
	public boolean equalsForLogin(User obj) {
		if(email.equals(obj.getEmail()) &&
password.equals(obj.getPassword())){
			return true;
		} else {
			return false;
		}
	}
	@Override
	public int hashCode() {
		final int prime = 31;
		int result = 1;
		result = prime * result + id;
		return result;
	}
	@Override
	public boolean equals(Object obj) {
		if (this == obj)
			return true;
		if (obj == null)
			return false;
		if (getClass() != obj.getClass())
			return false;
		User other = (User) obj;
		if (id != other.id)
			return false;
		return true;
	}
}
```

Código-fonte da classe UserDB:

```
public class UserDB extends ModelDBGeneric implements ModelDBView<User> {
	public UserDB() {
		super();
	}
	@Override
	public User get(Object id) throws ModelException {
```

```
			params = new ArrayList<>();
			params.add(id);
			return prepare(executeQuery(QueryDB.USER_SELECT, params));
		}
		@Override
		public void save(User m) throws ModelException {
			boolean exist = this.exist(m);
			params = new ArrayList<>();
			if (m.getId() == 0 || !exist) {
				params.addAll(m.toArrayWithoutId());
				executeUpdate(QueryDB.USER_INSERT, params);
			} else {
				params.addAll(m.toArrayIdLast());
				executeUpdate(QueryDB.USER_UPDATE, params);
			}
		}
		@Override
		public List<User> list() throws ModelException {
			return prepareAll(executeQuery(QueryDB.USER_SELECT_ALL));
		}
		@Override
		public void delete(Object id) throws ModelException {
			params = new ArrayList<>();
			params.add(id);
			executeUpdate(QueryDB.USER_DELETE, params);
		}
		@Override
		public boolean exist(User m) throws ModelException {
			try {
				params = new ArrayList<>();
				params.add(m.getId());

				ResultSet rs = executeQuery(QueryDB.USER_EXIST, params);
				if (rs.next() && rs.getInt(1) > 0) {
					return true;
				} else {
					return false;
				}
			} catch (SQLException e) {
				throw new ModelException(e);
			}
		}
		@Override
		public User prepare(ResultSet rs) throws ModelException {
			try {
				if (rs.next()) {
					return new User(rs.getInt("id"),
rs.getString("email"), rs.getString("password"), rs.getString("openid"),
rs.getBoolean("admin"), rs.getBoolean("authority"));
				}
			} catch (SQLException e) {
				throw new ModelException(e);
			}
			return null;
		}
		@Override
		public List<User> prepareAll(ResultSet rs) throws ModelException {
			List<User> list = null;
			try {
```

```
                    list = new ArrayList<>();
                    while (rs.next()) {
                          list.add(new User(rs.getInt("id"),
rs.getString("email"), rs.getString("password"), rs.getString("openid"),
rs.getBoolean("admin"), rs.getBoolean("authority")));
                    }
             } catch (SQLException e) {
                    throw new ModelException(e);
             }
             return list;
      }
      public User get(String email, String password) {
             // TODO Auto-generated method stub
             return null;
      }
}
```

Código-fonte da classe UserDBTest:

```
public class UserDBTest extends TestCase {
      private UserDB db;
      private User obj;
      public void testGet() throws ModelException {
             this.db = new UserDB();
             this.obj = this.db.get(2L);
             System.out.println(obj.toString());
      }
      public void testSave() throws ModelException {
             this.db = new UserDB();
             this.obj = new User(0, "emassssilsss@hotmail.com", "jkhkjhjkhj",
Util.generateHashMD5("jkhkjhjkhj"), true, false);
//           this.db.save(this.obj);

             this.obj = this.db.get(10L);
             this.obj.setEmail("email@mudou.com.br");
             this.db.save(this.obj);
      }
      public void testList() throws ModelException {
             this.db = new UserDB();
             List<User> list = this.db.list();
             for (User user : list) {
                    System.out.println(user.toString());
             }
      }
      public void testDelete() throws ModelException {
             this.db = new UserDB();
             this.db.delete(2L);
             this.db.delete(3L);
             this.db.delete(6L);
             this.db.delete(8L);
      }
}
```

Código-fonte da classe UserMock:

```
public class UserMock implements ModelDBView<User> {
      @Override
```

```
	public ResultSet executeQuery(QueryDB queryDB, List<Object> params)
throws ModelException {
		// TODO Auto-generated method stub
		return null;
	}
	@Override
	public ResultSet executeQuery(QueryDB queryDB) throws ModelException {
		// TODO Auto-generated method stub
		return null;
	}
	@Override
	public void executeUpdate(QueryDB queryDB, List<Object> params) throws
ModelException {
		// TODO Auto-generated method stub
	}
	@Override
	public User get(Object obj) throws ModelException {
		if(obj instanceof Integer){
			for (User u : DBMock.users) {
				if(u.getId() == ((int) obj)){
					return u;
				}
			}
		} else if (obj instanceof User){
			for (User u : DBMock.users) {
				if(u.equalsForLogin((User)obj)){
					return u;
				}
			}
		}
		return null;
	}
	@Override
	public void save(User m) throws ModelException {
		if(m.getId() == 0){
			m.setId(DBMock.users.size()+1);
			DBMock.users.add(m);
			return;
		}
		for (int i = 0; i < DBMock.users.size(); i++) {
			if(DBMock.users.get(i).getId() == m.getId()){
				DBMock.users.set(i, m);
				return;
			}
		}
		DBMock.users.add(m);
	}
	@Override
	public List<User> list() throws ModelException {
		return DBMock.users;
	}
	@Override
	public void delete(Object id) throws ModelException {
		for (User u : DBMock.users) {
			if(u.getId() == (int) id){
				DBMock.users.remove(u);
				break;
			}
		}
```

```
	}
	@Override
	public boolean exist(User m) throws ModelException {
		for (User u : DBMock.users) {
			if(u.equals(m)){
				return true;
			}
		}
		return false;
	}
	@Override
	public User prepare(ResultSet resultSet) throws ModelException {
		// TODO Auto-generated method stub
		return null;
	}
	@Override
	public List<User> prepareAll(ResultSet resultSet) throws
ModelException {
		// TODO Auto-generated method stub
		return null;
	}
}
```

Código-fonte da classe UserWService:

```
@RestController
@RequestMapping("/user")
public class UserWService extends RestWServiceGeneric<User>{
	private static final Logger log =
LoggerFactory.getLogger(UserWService.class);
	public UserWService() {
		super(new UserMock());
	}
	@RequestMapping("/get")
	public ResponseEntity<User> get(@RequestParam(value = "id") int id) {
		try {
			log.info("init");
			User obj = db.get(id);
			if (obj != null){
				log.info(obj.toString());
				return new ResponseEntity<User>(obj,
HttpStatus.OK);
			} else {
				return new ResponseEntity<>(HttpStatus.NOT_FOUND);
			}
		} catch (ModelException e) {
			log.error("Error: " + e.getMessage());
			return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
		}
	}
	@RequestMapping(value="/save",method=RequestMethod.POST)
	public ResponseEntity<Void> save(@RequestBody User m) {
		try {
			log.info("init");
			db.save(m);
			return new ResponseEntity<>(HttpStatus.OK);
		} catch (ModelException e) {
			log.error("Error: " + e.getMessage());
```

```
                  return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
            }
      }
      @RequestMapping("/list")
      public ResponseEntity<List<User>> list() {
            try {
                  log.info("init");
                  List<User> list = db.list();
                  if(list != null && list.size()>0){
                        return new ResponseEntity<List<User>>(list,
HttpStatus.OK);
                  } else {
                        return new
ResponseEntity<List<User>>(HttpStatus.NOT_FOUND);
                  }
            } catch (ModelException e) {
                  log.error("Error: " + e.getMessage());
                  return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
            }
      }
      @RequestMapping("/delete")
      public ResponseEntity<Void> delete(@RequestParam(value = "id") Long
id) {
            try {
                  log.info("init");
                  db.delete(id);
                  return new ResponseEntity<>(HttpStatus.OK);
            } catch (ModelException e) {
                  log.error("Error: " + e.getMessage());
                  return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
            }

      }
      @RequestMapping("/exist")
      public ResponseEntity<Boolean> exist(@RequestParam(value = "id") int
id) {
            try {
                  log.info("init");
                  return new ResponseEntity<>(db.exist(new User(id)),
HttpStatus.OK);
            } catch (ModelException e) {
                  log.error("Error: " + e.getMessage());
                  return new
ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
            }
      }
}
```

Código-fonte de configuração projeto Firebase:

```
      <!-- Firebase -->
      <script
src="https://www.gstatic.com/firebasejs/3.3.0/firebase.js"></script>
      <script>
        // Initialize Firebase
        var config = {
```

```
          apiKey: "AIzaSyBJsAqircZUErwTUBKVuySFAtf8_L9na0k",
          authDomain: "falasehhrio-4cf51.firebaseapp.com",
          databaseURL: "https://falasehhrio-4cf51.firebaseio.com",
          storageBucket: "falasehhrio-4cf51.appspot.com",
        };
        firebase.initializeApp(config);
      </script>
```

Código-fonte de redirecionamento Firebase:

```
(function () {
    'use strict';

    angular
        .module('app')
        .controller('LoginController', LoginController);

    LoginController.$inject    =    ['$location',    'AuthenticationService',
'UserService', 'FlashService'];
    function LoginController($location, AuthenticationService, FlashService)
{
        var vm = this;

        vm.login = login;
        vm.loginMedia = loginMedia;
        vm.loginMitreId = loginMitreId;

        (function initController() {
            console.log("reset login status");
            AuthenticationService.ClearCredentials();
        })();

        function login() {
            vm.dataLoading = true;
            AuthenticationService.Login(vm.username,   vm.password,   function
(response) {
                if (response.success) {
                    AuthenticationService.SetCredentials(vm.username,
vm.password);
                    $location.path('/');
                } else {
                    FlashService.Error(response.message);
                    vm.dataLoading = false;
                }
            });
        };

             function loginMedia(media) {
               vm.dataLoading = true;

               console.log("[START signout]");
               firebase.auth().signOut();
               console.log("[END signout]");

               console.log("[START createprovider]");
          var provider = new firebase.auth.GoogleAuthProvider();
          console.log("[END createprovider]");
```

```
          console.log("[START addscopes]");
          provider.addScope('https://www.googleapis.com/auth/plus.login');
          console.log("[END addscopes]");

          console.log("[START signin]");
          firebase.auth().signInWithRedirect(provider);
          console.log("[END signin]");
             };

        function loginMitreId(media) {
            console.log(media);
        };

        /**
         * initApp handles setting up UI event listeners and registering
Firebase auth listeners:
         *   - firebase.auth().onAuthStateChanged: This listener is called
when the user is signed in or
         *     out, and that is where we update the UI.
         *   - firebase.auth().getRedirectResult(): This promise completes
when the user gets back from
         *     the auth redirect flow. It is where you can get the OAuth
access token from the IDP.
         */
        function initApp() {
          console.log("Result from Redirect auth flow.");
          console.log("[START getidptoken]");
          firebase.auth().getRedirectResult().then(function(result) {
            if (result.credential) {
              console.log("This gives you a Google Access Token. You can use
it to access the Google API.");
              var token = result.credential.accessToken;
              console.log("[START_EXCLUDE]");
              vm.token = token;
              console.log("[START_SETCREDENTIALS]");
              AuthenticationService.SetCredentials("google",
result.user.email, token);
              console.log("[END_SETCREDENTIALS]");
              console.log("[START_RELOAD]");
              $location.path('/');
              console.log("[END_RELOAD]");
            } else {
             vm.token = 'null';
              console.log("[END_EXCLUDE]");
            }
            console.log("The signed-in user info.");
            vm.user = result.user;
          }).catch(function(error) {
            console.log("Handle Errors here.");
            var errorCode = error.code;
            var errorMessage = error.message;
            console.log("The email of the user's account used.");
            var email = error.email;
            console.log("The  firebase.auth.AuthCredential  type  that  was
used.");
            var credential = error.credential;
            console.log("[START_EXCLUDE]");
            if     (errorCode     ===     'auth/account-exists-with-different-
credential') {
```

```
              alert('You have already signed up with a different auth
provider for that email.');
              console.log("If you are using multiple auth providers on your
app you should handle linking");
              console.log("the user's accounts here.");
            } else {
              console.error(error);
            }
            console.log("[END_EXCLUDE]");
          });
          console.log("[END getidptoken]");
          console.log("Listening for auth state changes.");
          console.log("[START authstatelistener]");
          firebase.auth().onAuthStateChanged(function(user) {
            if (user) {
             console.log("[START_SETCREDENTIALS]");
                AuthenticationService.SetCredentials("google",    user.email,
vm.token);
                console.log("[END_SETCREDENTIALS]");
                console.log("[START_RELOAD]");
                $location.path('/');
                console.log("[END_RELOAD]");
            } else {
                vm.dataLoading = false;
            }
          });
        }

        initApp();
    }

})();
```