UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA

FELIPE AUGUSTO ZBOROWSKI

RAFAELA SOMAVILA LIMA

SGCI – SISTEMA DE GERENCIAMENTO DE CASAS INTELIGENTES

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA
2017

FELIPE AUGUSTO ZBOROWSKI
RAFAELA SOMAVILA LIMA

SGCI – SISTEMA DE GERENCIAMENTO DE CASAS INTELIGENTES

Trabalho de Conclusão de Curso de Bacharelado em Sistemas de Informação do Departamento Acadêmico de Informática – DAINF - da Universidade Tecnológica Federal do Paraná - UTFPR, como requisito parcial para obtenção do título de "Bacharel em Sistemas de Informação".

Orientadora: Profª. Dra. Ana Cristina Kochem Vendramin
Coorientadora: Profª. Drª. Viktória Zsók

CURITIBA
2017

Ministério da Educação
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
Câmpus Curitiba
Diretoria de Graduação e Educação Profissional
Departamento Acadêmico de Informática
Coordenação do Curso de Bacharelado em Sistemas de Informação

TERMO DE APROVAÇÃO

"SGCI – SISTEMA DE GERENCIAMENTO DE CASAS INTELIGENTES"

por

"Felipe Augusto Zborowski e Rafaela Somavila Lima"

Este Trabalho de Conclusão de Curso foi apresentado no dia **22** de **junho** de **2017** como requisito parcial à obtenção do grau de Bacharel em Sistemas de Informação na Universidade Tecnológica Federal do Paraná - UTFPR - Câmpus Curitiba. O(a)(s) aluno(a)(s) foi(ram) arguido(a)(s) pelos membros da Banca de Avaliação abaixo assinados. Após deliberação a Banca de Avaliação considerou o trabalho _____.

_____ **<Prof. Ana Cristina Kochem Vendramin >** (Presidente - UTFPR/Curitiba)	_____ **<Prof. Luiz Augusto Pelisson>** (Avaliador 1 – UTFPR/Curitiba)
_____ **<Prof. Wilson Horstmeyer Bogado>** (Avaliador 2 – UTFPR/Curitiba)	_____ **<Prof. Leyza Baldo Dorini >** (Professor Responsável pelo TCC – UTFPR/Curitiba)
_____ **<Prof. Leonelo Dell Anhol Almeida >** (Coordenador(a) do curso de Bacharelado em Sistemas de Informação – UTFPR/Curitiba)	

"A Folha de Aprovação assinada encontra-se na Coordenação do Curso."

AGRADECIMENTOS

Agradecemos às professoras Drª. Ana Cristina Kochem Vendramin e Drª Viktória Zsók, orientadora e coorientadora, pela orientação em todas as etapas do desenvolvimento deste projeto.

Agradecemos aos nossos pais, amigos e demais familiares que nos incentivaram e nos apoiaram em todos os momentos.

E agradecemos a todos que, de maneira direta ou indireta, contribuíram para nossa formação pessoal e acadêmica.

"*The trouble is, you think you have time*."

Jack Kornfield

RESUMO

ZBOROWSKI, Felipe Augusto; LIMA, Rafaela Somavila. SGCI - SISTEMA DE GERENCIAMENTO DE CASAS INTELIGENTES. 63 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2017.

A tecnologia está cada vez mais presente nos diferentes momentos do dia a dia da sociedade. É possível notar cada vez menos a existência de tempo realmente livre diariamente, tendo em vista que esse tempo é consumido muitas vezes entre atividades simples em casa que muitas vezes poderiam ser automatizados pela tecnologia, por exemplo trancar portas e controlar temperatura. Neste projeto é apresentada uma solução para automatização residencial que tem o intuito de facilitar, economizar tempo na realização de tarefas e aumentar a comodidade do morador no dia a dia de uma casa. Além desses pontos, uma casa automatizada também contribui para economia de energia, segurança e pode auxiliar portadores de necessidades especiais a realizarem tarefas. O conceito apresentado de *smart house* permite que o morador possa poupar o tempo anteriormente gasto em tarefas simples e repetitivas, mas que consomem a longo prazo uma boa porção do seu dia. Essa tecnologia permite que um simples toque no celular, ou apenas uma mudança de temperatura, acionem dispositivos na moradia para manter tudo como o desejado pelo usuário, permitindo assim que este não tenha de realizar essas tarefas e possa confortavelmente seguir com suas demais atividades. Em outra frente, o projeto também demonstra o desenvolvimento do sistema em uma linguagem funcional relativamente pouco conhecida, a *Clean*, demonstrando como tecnologias pouco conhecidas podem se tornar ferramentas para solucionar problemas.

Palavras-chave: Dispositivos móveis, casa inteligente, *Clean*, automatização.

ABSTRACT

ZBOROWSKI, Felipe Augusto; LIMA, Rafaela Somavila. SHMS - SMART HOUSE MANAGEMENT SYSTEM. 63 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática, Universidade Tecnológica Federal do Paraná. Curitiba, 2017.

Technology is increasingly present in the different moments of everyday life of society; it is possible to notice the decreasing amount of free time, considering that this time is often consumed among simple activities at home that could be often automated by technology, for example locking doors and controlling temperature. This project presents a solution for residential automation that aims to facilitate, to save time while performing tasks and to help in the routine of a household. Besides these points, an automated house also contributes to energy saving, security and can help a disabled person to perform tasks. The presented concept of smart house allows the resident to save the time previously spent on simple and repetitive tasks that consume a good portion of the day. This technology allows a simple touch on the cell phone, or a change in temperature to trigger devices in the house to keep everything as desired by the user, so that the user does not have to perform these tasks and can comfortably follow with his other activities. The project also demonstrates the development of the system in a relatively little-known functional language: *Clean*, demonstrating how unknown technologies can become problem-solving tools.

Keywords: Mobile devices, smart house, *Clean*, automation.

LISTA DE FIGURAS

LISTA DE SIGLAS

API	*Application Programming Interface*
IDE	*Integrated Development Environment*
IP	*Internet Protocol*
JDK	*Java Development Kit*
SDK	*Software Development Kit*

SUMÁRIO

1 INTRODUÇÃO

A ideia de *Smart Home* (Casa Inteligente, em tradução livre) vem de uma tendência tecnológica para habitações futuras que utiliza uma casa como plataforma e adiciona periféricos que a automatizam e deixam-na inteligente (ZHANG et al., 2011). Em uma casa inteligente é possível integrar tecnologias digitais à mesma e fornecer um estilo de vida seguro, confortável e conveniente para o morador (WANG et al., 2005). Conceitualmente, uma casa inteligente deve realizar diversas tarefas de forma autônoma e independente da ação de seus ocupantes. Alguns exemplos de funções passíveis de realização automática vão desde coisas simples como trancar uma porta, ligar o ar-condicionado e controlar cortinas até ações mais complexas de controle do ambiente (ALAM et al., 2012).

O propósito de uma casa inteligente é controlar vários periféricos com um único gerenciador (ZHANG et al., 2011), seja por meio de uma aplicação de celular, aplicação web ou até mesmo por reconhecimento de voz.

O ramo de automatização de casas, por mais que seja uma área de grande interesse por parte da indústria e também da população em busca de uma moradia mais segura e confortável, ainda não apresenta equipamentos com preços acessíveis para que de fato pessoas possam utilizar essa tecnologia. Automatizar uma residência completamente pode chegar a um preço final mais alto que o próprio imóvel originalmente. É importante lembrar que além dos custos com a compra dos dispositivos, existem outros que estão implícitos (ROSE, 2001), como por exemplo eventuais modificações que necessitem ser feitas para instalação, como montagem dos periféricos e configuração de *software* de gerenciamento, além da manutenção dos equipamentos.

Outro ponto a ser considerado sobre a situação atual do ramo de casas inteligentes é o fato de que os equipamentos disponíveis hoje no mercado não possuem um padrão, ou seja, alguns periféricos podem não funcionar corretamente em conjunto com outros (ROSE, 2001).

Este trabalho apresenta um sistema de gerenciamento e controle de uma casa inteligente, tendo como primeira premissa a simulação de ocorrências referentes aos sensores internos ou externos de uma casa para a geração de eventos de controle sobre as diversas unidades da casa, como por exemplo, portas, ar-condicionado e aquecedores.

O projeto do sistema em si conta com a elaboração de um plano de projeto distribuído que controle individualmente, e ainda de forma integrada, as diversas partes da casa, como fechaduras, janelas, luzes, entre outros, e busca apresentar tudo isso em um modelo teórico funcional, trabalhando como uma prova de conceito para aplicação física futura.

A linguagem escolhida para a implementação do software, por ser mais orientada a eventos do que precisamente a objetos ou classes, chama-se *Clean* (PLASMEIJER; EEKELEN, 2001). Apesar desta ser uma linguagem pouco usada e não ter uma documentação completa acessível, acredita-se que se trate de uma boa opção para este trabalho, uma vez que esta, além de apresentar uma prova de conceito para o caso das casas inteligentes, também criará nova documentação acadêmica e técnica para uma linguagem de programação pouco conhecida.

1.1 JUSTIFICATIVA

A escolha do tema deste projeto foi dada com base em uma necessidade recorrente da vida contemporânea, em que cada vez mais as pessoas possuem menos tempo para realizarem atividades corriqueiras do dia a dia que poderiam facilmente ser automatizadas com a ajuda de computadores, criando uma forma mais cômoda de se gerir um ambiente, utilizando-se de apenas um aplicativo ou dispositivo, ao invés de vários. A tecnologia aqui apresentada possibilita um custo baixo para a implementação do servidor, necessitando que este utilize apenas um computador simples com conexão à rede sem fio da residência.

A escolha da linguagem Clean, apesar de um tanto quanto desconhecida e de difícil acesso, deu-se pelos seguintes motivos: simplicidade da linguagem em executar funções; ser uma linguagem orientada a tarefas e não a objetos, o que permite maior liberdade de criação e distribuição de ações pelo sistema; os autores deste trabalho terem iniciado a idealização do projeto durante o estágio realizado em 2016, durante o período de intercâmbio, na Universidade Eötvös Loránd, sediada em Budapeste, Hungria, sob a supervisão da coorientadora Viktória Zsók. Uma vez que o projeto já estava com seus requisitos definidos, foi mais interessante continuar o projeto seguindo sua proposta inicial que contempla o uso da linguagem *Clean*. Um desafio que foi proposto pela escolha da linguagem *Clean* foi a criação de uma documentação sobre esta linguagem para pesquisas futuras.

Somando esses fatores de escolha, acredita-se que o presente projeto possa trazer um bom avanço tanto no referencial para as pesquisas sobre casas inteligentes quanto para futuros projetos utilizando-se da linguagem Clean.

1.2 OBJETIVO GERAL

O objetivo geral do presente trabalho é desenvolver um *software* de gerenciamento para uma casa inteligente, por meio do qual o usuário poderá monitorar e configurar componentes em diferentes áreas da casa. As áreas de foco são as seguintes: iluminação, trancas, temperatura, plugues de energia, cortinas e também sensores e monitoramento sobre as pessoas dentro do ambiente.

1.3 OBJETIVOS ESPECÍFICOS

Os objetivos específicos do presente trabalho são:

- Desenvolver um *software* funcional utilizando-se da linguagem de programação *Clean*;
- Desenvolver um aplicativo para celular que auxilie a controlar as funções de uma casa;
- Contribuir com a documentação da linguagem *Clean*.

1.4 ORGANIZAÇÃO DO DOCUMENTO

O presente documento encontra-se organizado da seguinte forma. O primeiro capítulo introduz de forma simples o conceito de casas inteligentes e apresenta a linguagem de programação utilizada na implementação do sistema, também apresenta a justificativa do projeto e os objetivos gerais e específicos do mesmo. O segundo capítulo apresenta o levantamento bibliográfico, importante para o embasamento e fundamentação do projeto. A metodologia, descrevendo as etapas de desenvolvimento do projeto é apresentada no terceiro capítulo. O quarto capítulo apresenta tópicos sobre o desenvolvimento do sistema servidor e do aplicativo móvel, tais como funcionamento e uso do sistema e telas do aplicativo. As conclusões

sobre o desenvolvimento do projeto são apontadas no quinto capítulo. Por fim, a documentação da linguagem Clean encontra-se no Apêndice.

2 LEVANTAMENTO BIBLIOGRÁFICO

Esta seção apresenta uma breve introdução a *Smart Homes*, à linguagem utilizada para desenvolvimento do sistema e demais tópicos relacionados à elaboração do projeto.

2.1 CASAS INTELIGENTES

Com o avanço tecnológico e a expansão dos serviços, a internet caminha em direção à Internet das Coisas (*Internet of Things*). A Internet das Coisas é a internet na qual a rede já existente irá se conectar a objetos do mundo real, tais como veículos, eletrodomésticos e máquinas industriais (GAIKWAD et al., 2015), sendo que o controle desses objetos pode ser feito automaticamente ou à distância (BING et al., 2011), através de um dispositivo móvel, por exemplo.

Casas inteligentes podem ser definidas como um ambiente automatizado interconectado no qual as tarefas que são repetidas frequentemente podem ser feitas de forma automática, trazendo mais conforto e segurança para seus moradores, pois também permite o controle e monitoramento da casa à distância. Para ser considerada inteligente, a casa deve ser equipada com diversos sistemas que devem se comunicar entre si (KAVITHA, 2009), por exemplo, eletrodomésticos inteligentes e sensores (ÖZKAN; AYBAR, 2011). Dentre as tarefas que podem ser automatizadas, pode-se destacar o controle de luminosidade e temperatura (ALAM et al., 2012), controle de eletrodomésticos como cafeteiras, fogões e ar-condicionado (ÖZKAN; AYBAR, 2011) e também controle de fechaduras e portas.

Atualmente disponíveis no mercado, existem aparelhos que funcionam como mediadores para esses produtos inteligentes que se interconectam, entre eles o *Amazon Echo* (AMAZON, 2016b), o *Apple HomeKit* (APPLE, 2016), o *Google Home* (GOOGLE, 2016), o *Samsung SmartThings* (SAMSUNG, 2017) e o *Wink* (WINK, 2017).

O *Amazon Echo* (AMAZON, 2016b) funciona como um gerenciador central voltado principalmente para permitir ações controladas por voz, quando se conecta ao *Alexa Voice Service* (AMAZON, 2016a), também propriedade da *Amazon*, permitindo ao usuário tocar músicas, pedir informações sobre notícias, esportes, pedir indicações de restaurantes e comprar produtos em lojas *online*. Este produto ainda tem a capacidade de interagir com outros dispositivos independentes, de modo a pedir que estes realizem suas funções, permitindo assim

uma automatização da casa, porém com a necessidade de diversos produtos e *softwares* diferentes.

O *Apple HomeKit* (APPLE, 2016) é um aplicativo para celular e *tablet* que permite que o usuário possa controlar diversos "acessórios" relacionados à casa inteligente, como luzes, fechaduras, tomadas, câmeras e ventiladores. O aplicativo permite que o usuário crie cenários com os acessórios, que fará as ações necessárias de forma automática, um exemplo é um cenário chamado "Sair de Casa", que apaga luzes e tranca portas. Apenas produtos que sejam compatíveis com o *HomeKit* (em sua maioria produtos da própria Apple) podem ser inseridos nesses cenários e controlados utilizando-se do aplicativo.

O *Google Home* (GOOGLE, 2016) funciona de forma bastante parecida com o *Amazon Echo* (AMAZON, 2016b), onde um aparelho é instalado para detectar comandos de voz e responder a tais comandos, novamente permitindo utilizar-se de servidores de música e receber respostas de pesquisas no *Google*. Com estruturas inteligentes de *software*, o aparelho aprende os hábitos dos usuários para a criação de perfis. O produto tem ainda a capacidade de controlar os demais dispositivos inteligentes presentes na casa, como *Smart TVs*, assim como controladores de luz, termostato e tomadas.

O *Samsung Smartthings* (SAMSUNG, 2017) também é um aparelho central que trabalha em conjunto com um aplicativo de celular, ele se conecta com outros aparelhos inteligentes da residência de diversas marcas, como fechaduras, lâmpadas e câmeras, e também permite criar rotinas para que o aparelho comande ações para realização de tarefas. Esse aparelho pode ser conectado com o *Amazon Echo* e o *Google Home* para que o morador possa dar comandos através da fala.

Por fim, o *Wink* (WINK, 2017) é um aplicativo para celular que auxilia no controle dos periféricos inteligentes presentes na casa, ele funciona com uma grande quantidade de marcas, entre elas *GE*, *Phillips*, e o *Alexa Voice Service* (AMAZON, 2016a). Assim como os outros aparelhos citados anteriormente, o *Wink* permite criar atalhos para facilitar o dia a dia dos usuários, desligando luzes à noite, ou ligando a cafeteira pela manhã.

2.2 CLEAN

Clean (PLASMEIJER; EEKELEN, 2001) é uma linguagem de programação funcional, de fácil otimização, criada em 1987 pelo Grupo de Pesquisa de Tecnologia de

Software da Universidade de Radboud de Nimega, na Holanda e ainda está em desenvolvimento.

Por se tratar de uma linguagem funcional, Clean trabalha com a utilização de funções que devem ser definidas durante a elaboração do sistema. O programa então quando é executado busca a partir da função de início as ordens de como prosseguir baseado nas funções que foram criadas, gerando uma expressão a ser avaliada pelo compilador (KOOPMAN et al., 2002). O sistema segue uma ordem de compilação baseada em só trabalhar as funções uma vez que essas são requisitadas pela ordem de utilização, uma técnica conhecida como *lazy compiling*.

Em *Clean* existe uma expressão chamada "*Start*", essa expressão é avaliada inicialmente quando um programa é executado (KOOPMAN et al., 2002). A biblioteca padrão utilizada *Standard Environment* é definida como *StdEnv*. Uma característica da linguagem é que o programa deve ter o mesmo nome do arquivo no qual se está trabalhando.

A Figura 1 apresenta um exemplo de como calcular uma soma de dois números, ilustrando a expressão "*Start*" e a definição da biblioteca *StdEnv*.

```
1  module exemploSoma
2  import StdEnv
3
4  Start = 5+3
5
```

Figura 1: Exemplo de código de soma em *Clean*.
Fonte: Autoria própria.

Alterando-se o exemplo de soma, é possível definir uma função, como demonstrado na Figura 2:

```
1  module exemploSoma2
2  import StdEnv
3
4  Soma:: Int Int -> Int
5  Soma x y = x + y
6
7  Start = Soma 5 3
```

Figura 2: Exemplo de definição de função em *Clean*.
Fonte: Autoria própria.

Ao definir uma função, o programa a utilizará do mesmo modo como emprega as funções que são definidas pela biblioteca padrão. Definições de funções são sempre parte de um *module*, e esse *module* é sempre armazenado em um arquivo (KOOPMAN et al., 2002).

Clean possui um método de avaliação chamado preguiçoso (*lazy*). Nesse tipo de avaliação, uma expressão ou parte dela somente é calculada quando realmente for necessária sua utilização pelo programa (PLASMEIJER; EEKELEN, 2001).

2.3 ITASKS

O *iTask System* (*iTasks*) é um conjunto de ferramentas complementar ao *Clean*, que permite que se crie um fluxo de trabalho, utilizando-se de linguagem funcional, com um nível alto de abstração. Os sistemas de fluxo são um modo automatizado de coordenar tarefas que devem ser executadas (PLASMEIJER et al., 2008).

No caso deste projeto, o *iTasks* é acessado pelo navegador. A Figura 3 mostra como a tela de autenticação do servidor é apresentada para o usuário em uma aplicação padrão quando o servidor é acessado inicialmente, requisitando ao usuário a entrada de seu nome de usuário e sua senha.

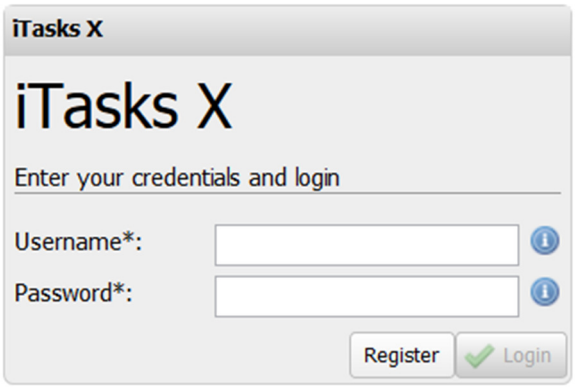

Figura 3: Tela de autenticação do *iTasks*.
Fonte: Autoria própria.

A Figura 4 ilustra um dos *workflows* criados pelo *iTasks* para a demonstração de dados armazenados pelo servidor. Sendo nesse caso a apresentação dos dados presentes no servidor sobre o estado dos sensores, aqui apresentando o tipo dos sensores adicionados ao servidor, e

também o valor recebido por cada um deles e uma *flag* indicando se o servidor já trabalhou com aquela informação específica recebida.

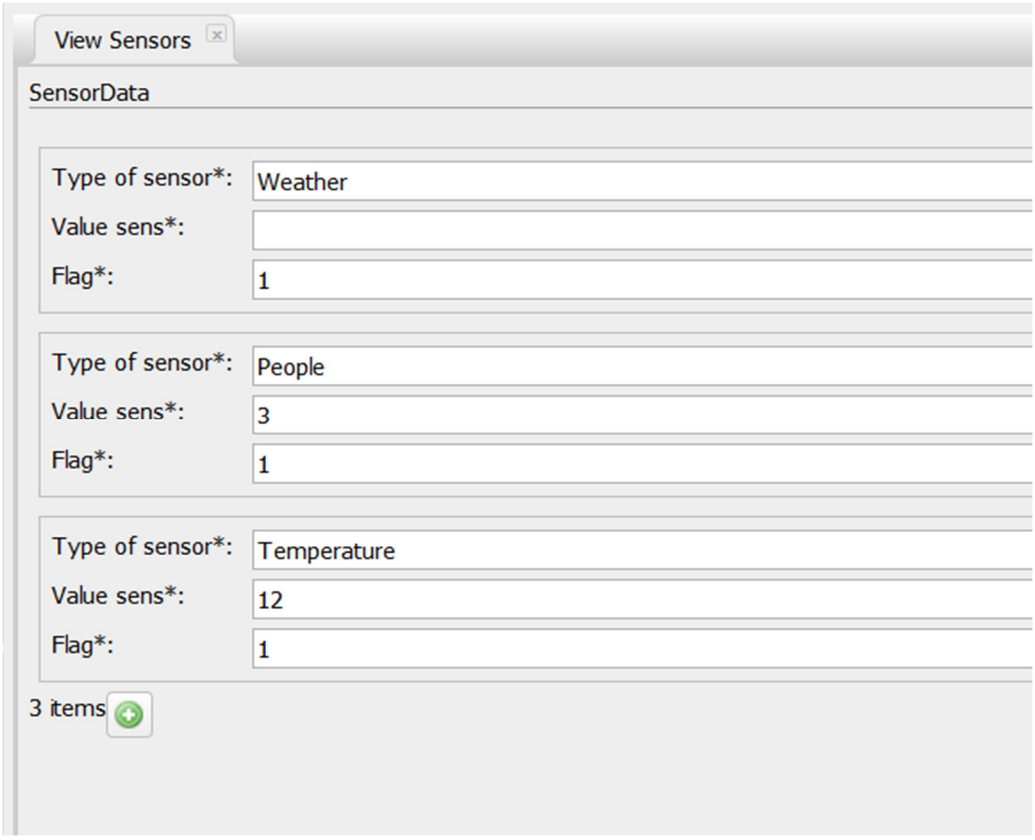

Figura 4: Exemplo de workflow com o *iTasks*.
Fonte: Autoria própria.

3 METODOLOGIA

Neste capítulo apresenta-se a metodologia utilizada a fim de alcançar os objetivos do projeto. Para tal, apresenta-se de que forma o desenvolvimento será abordado e, em seguida, as tecnologias a serem utilizadas, tais como a linguagem de programação e o ambiente de desenvolvimento.

3.1 ABORDAGEM

Partindo da ideia inicial de desenvolver um *software* que simulasse uma casa inteligente, o passo seguinte consistiu na realização de uma pesquisa bibliográfica, buscando principalmente por artigos sobre trabalhos relacionados ao tema de automatização de habitações para, então, elaborar a revisão da literatura.

O desenvolvimento do *software* e do aplicativo para *smartphone* foram realizados utilizando-se das linguagens de programação *Clean* e Java, respectivamente.

3.2 TECNOLOGIAS

Na presente seção são abordadas as tecnologias de desenvolvimento utilizadas para a elaboração dos produtos de *software* que constituem o projeto, sendo eles o desenvolvimento do servidor na linguagem *Clean* e o aplicativo móvel desenvolvido para uso em dispositivos Android.

3.2.1 Ambientes de Programação

O projeto tem seu desenvolvimento de *softwares* dividido em duas diferentes frentes de trabalho, uma tratando-se do servidor de controle, desenvolvido com a utilização da linguagem funcional denominada *Clean* (NÖCKER et al., 1991). Nessa parte do desenvolvimento os *softwares* utilizados em sua criação foram a plataforma de desenvolvimento *Clean* IDE[1] com a adição do ambiente programacional denominado *iTasks*[2],

[1] *Clean* IDE. Disponível em: http://clean.cs.ru.nl/Clean
[2] *iTasks*. Disponível em: http://clean.cs.ru.nl/ITasks

todos os programas necessários para o desenvolvimento podem ser adquiridos por *download* em seus respectivos endereços *web* sem necessidade de pagamentos de licença ou aquisição.

Na frente de desenvolvimento para dispositivos móveis, especificamente um aplicativo Android que age como cliente do servidor em *Clean*, fez-se uso da IDE oficial disponibilizada no site de desenvolvimento Android[3] que é o *software Android Studio*. Juntamente com a IDE instalada, algumas ferramentas de desenvolvimento adicionais são obtidas no pacote, como o SDK (*Software Development Kit*) e versões do Android para serem emuladas para realização de testes do aplicativo.

De acordo com (SIMS, 2014), a linguagem padrão para desenvolvimento Android é Java. Também se faz necessário para a utilização do *Android Studio* que se tenha instalado o JDK (*Java Development Kit*) que está disponível no site da Oracle[4].

Para os quesitos de teste do aplicativo não foi utilizado nenhum *software* específico, uma vez que se fez mais simples e acessível de modo que os testes fossem realizados diretamente no aparelho móvel desejado, posto que o processamento do mesmo é mais indicado para o uso dos aplicativos do que a utilização de um emulador dentro de um dispositivo diferente.

3.2.2 Linguagem de Programação

Para a criação do *software* de controle, foi utilizada a linguagem de programação *Clean* (NÖCKER et al., 1991). Essa linguagem, apesar de ser pouco conhecida e utilizada, faz-se válida para as implementações desejadas do *software*, dadas suas características de programação funcional. É necessária a elaboração e simulação de diversos elementos, por exemplo os sensores e a aparelhagem da casa, e esses dados gerados para os testes do *software* de controle podem ser projetados também em *Clean* ou em qualquer outra linguagem, pois são apenas tratados como ferramentas para os testes.

Já para o aplicativo móvel, como este foi desenvolvido para Android, a linguagem de programação padrão indicada pelo próprio Google, desenvolvedor do sistema Android, é a linguagem Java. Esta será utilizada junto das APIs (*Application Programming Interface*)

[3] Pode ser acessado em: https://developer.android.com
[4] Disponível para acesso em: http://www.oracle.com/technetwork/java/javase/

específicas do Android SDK (*Software Development Kit*) para o desenvolvimento da arquitetura e aplicativo móvel desejado.

4 DESENVOLVIMENTO DAS APLICAÇÕES

Este capítulo apresenta uma explicação sobre como tanto o *software* de controle quanto o aplicativo móvel foram desenvolvidos. Na primeira seção, disserta-se sobre o funcionamento do servidor. Na sequência, a segunda seção apresenta o funcionamento do aplicativo móvel.

4.1 *SOFTWARE* SERVIDOR

O sistema objeto deste estudo tem seu funcionamento baseado no uso de reatores e eventos; isso é, cada vez que um novo dado é recebido por algum dos sensores simulados pelo sistema, o mesmo é tratado e analisado. O objetivo dessa análise é definir se alguma ação deve ser tomada, no caso alguma de suas condições base, sejam elas as definidas pelo usuário nas preferências ou as definidas no padrão do sistema. Após isso, é criado um tipo de evento para ser trabalhado nos reatores, para que algum dos componentes exteriores seja alertado e trabalhado.

Os sensores implementados até o presente estado do sistema são os sensores de:

- Pessoas: deve saber o número de indivíduos no interior da casa;
- Temperatura: mantém o trabalho de um termostato de ter a temperatura atual no interior dos ambientes da moradia;
- Clima: com a função de definir se está havendo chuva no exterior;
- Luzes: deve manter os dados sobre a iluminação do ambiente.

Na Figura 5 tem-se um exemplo de um receptor de sensor, neste caso o de clima. Caso o sensor tenha recebido um valor novo, ainda não tratado, no sistema determinado pelo número 0 em sua *flag*, de forma binária, um novo método é chamado para tratar esse evento.

A função se inicia com a definição do tipo de sensor do qual os dados são esperados e a inicialização de um loop com a função de esperar que um dado daquele tipo que ainda não tenha recebido tratamento nessa informação (Linha 1). Após o sistema receber toda informação dos sensores (Linha 2), ele deve salvar o sensor indicado em uma nova variável (Linha 3), e

fazer a remoção (Linha 4) do mesmo da lista de sensores ativos, apenas para o reinserir com o método adequado dependendo dos valores nele salvos (Linhas 5 e 6).

```
1 |senType == "Weather" = forever (wait "" (\senPool -> waitForSen senPool senType 0) SensorData
2  >>| get SensorData >>=
3  \senPool -> return (getFromPoolS senPool senType) >>=
4  \sen -> upd (\t -> remSensor t sen) SensorData >>| if(sen.valueSens == 1)
5  ( flagSens sen >>| isRaining >>| return () )
6  ( flagSens sen >>| return() ) )
7  @! ()
```

Figura 5: Receptor dos sensores aguardando que os dados dos sensores sejam atualizados.
Fonte: Autoria própria.

Na Figura 6 pode-se ver um método que cria efetivamente um evento, alertando os reatores de que esses devem tomar uma ação caso o seu tipo de evento tenha sido criado. Cada reator é mantido funcionando de forma contínua, vasculhando uma pilha de eventos buscando por aqueles eventos que tenham o mesmo tipo que ele, a Figura 7 ilustra esse processo.

```
1 isRaining = get EventPool >>= \evPool ->
2     (set (evPool++[{mytype = "Generic", message = "raining",
3       componentId = "WeatherSensor"}]) EventPool)
```

Figura 6: Exemplo de evento criado por um receptor para ser enviado aos reatores.
Fonte: Autoria própria.

```
1 | evType == "Locks" =   forever (wait "" (\evPool -> waitForE evPool evType) EventPool
2 >>| get EventPool >>=
3 \evPool -> return (getFromPoolE evPool evType) >>=
4 \ev -> upd (\t -> remFromPool t ev) EventPool >>|
5 doors {mytype = ev.mytype,
6 message = toString (drop 1 (dropWhile (\x = x <> '|') (fromString ev.message))),
7 componentId = toString (takeWhile (\x = x <> '|') (fromString ev.message))  } 1)
8 @! ()
```

Figura 7: Exemplo de reator que deve esperar eventos do mesmo tipo aparecerem.
Fonte: Autoria própria.

Os reatores atualmente disponíveis no sistema, são os seguintes:

- Genérico: trata de eventos mais gerais, que influenciam um ou mais dos demais reatores, eventos desses reatores são geralmente aqueles criados pelos receptores

dos sensores, como por exemplo avisos se está ou não chovendo, de que o dia está muito quente, ou que a casa está vazia;

- Tomadas: trata eventos decorrentes da necessidade de habilitar ou desabilitar uma certa saída de energia na casa;
- Aquecimento: esse reator recebe eventos relacionados ao controle de temperatura dos ambientes, e envia ordens para os aparelhos de aquecimento, como ar-condicionado e aquecedores, de forma automatizada;
- Aquecimento manual: como a sua contraparte citada anteriormente, esse reator tem por função novamente o controle de aparelhos de controle de temperatura, mas sendo totalmente dedicado ao controle direto pelo usuário;
- Luzes: o reator que controla tanto as funções das lâmpadas quanto das persianas na casa, permitindo acender ou apagar, tal como abrir e fechar, no caso das persianas, para maior controle do grau de luminosidade disponível no ambiente.

4.1.1 Estruturação de Dados

O funcionamento do sistema é baseado em dados compartilhados dentro deste, e separados em diversos tipos diferenciados de dado, sendo cada tipo trabalhado por uma diferente estrutura.

A estrutura apresentada na Figura 8 ilustra cada tipo de dado de forma genérica.

```
1  :: MyEvent = { mytype :: String
2               ,message :: String
3               ,componentId :: String
4               }
5
6  :: Components = {    component  :: String
7                   ,value      :: String
8                   ,state      :: String
9                   ,room       :: String
10                 }
11
12
13 :: Sensor = { typeOfSensor :: String
14              ,valueSens :: Int
15              , flag :: Int}
16
17
18 :: Preferences = { typeComp :: String
19                  ,valueTurnOn :: Int
20                  ,valueSet :: Int
21                  ,valueTurnOff :: Int}
```

Figura 8: Exemplo de estrutura dos dados de forma genérica.
Fonte: Autoria própria.

Os tipos de dados são:

- Eventos: uma estrutura de dados com objetivo de regular os diversos eventos que podem causar mudanças no estado de objetos no interior da casa. Essa estrutura de dados é normalmente criada pelos receptores dos sensores ou pelo reator genérico e tratada pelos demais reatores. As variáveis criadas no interior dessa estrutura são o seu tipo, mensagem e identificação de componente que originou esse evento. A estrutura de dados "Eventos" é demonstrada na Figura 8 (Linhas 1 a 4). Um exemplo de evento criado é visto na Figura 6 onde é criado um evento do tipo Genérico, para alertar a existência de precipitação, enviado pelo sensor de clima, evento esse criado com as variáveis de tipo "*Generic*", mensagem "isR*aining*" e com a identificação

de componente "*WeatherSensor*". Na Figura 9 demonstra-se a estrutura desse mesmo evento;

```
1 {   "mytype":"Generic",
2     "message":"isRaining",
3     "componentId":"WeatherSensor"}
```

Figura 9: Exemplo de estrutura de evento.
Fonte: Autoria própria.

- Componentes: tratam de uma representação dos objetos físicos presentes na casa, como lâmpadas, tomadas e aquecedores. Esses dados são afetados por métodos dentro do sistema, que são chamados pelos reatores quando estes se fazem necessários. Suas características são o nome do componente, valor que ele está trabalhando, estado e qual ambiente da casa ele ocupa. Estrutura demonstrada na Figura 8 (Linhas 6 a 10). Na Figura 10 temos um exemplo de componente demonstrando a identificação de um aparelho de ar condicionado, ligados a 10ºC;

```
1 {   "component":"AC",
2     "value":"10",
3     "state":"1",
4     "room":"1"  }
```

Figura 10: Componente demonstrando a identificação de ar condicionado ligado.
Fonte: Autoria própria.

- Sensores: estrutura responsável pela recepção dos dados dos sensores, os valores salvos nessa estrutura são analisados pelos receptores e reenviados, se necessário, em forma de evento para serem trabalhados pelos reatores. Os dados contidos nessa estrutura são o tipo de sensor, o valor que este está sinalizando e uma *flag* binária com o intuito de permitir ao receptor marcar se aquele dado já foi lido e trabalhado. A estrutura de dados "Sensores" está demonstrada na Figura 8 (Linhas 13 a 15). Um exemplo de dados salvos em um sensor pode ser visto na Figura 11, que demonstra os valores recebidos de um sensor de presença na residência, alertando a presença

de 3 pessoas, e demonstrando em sua *flag* que as ações relativas a essa informação já foram tomadas;

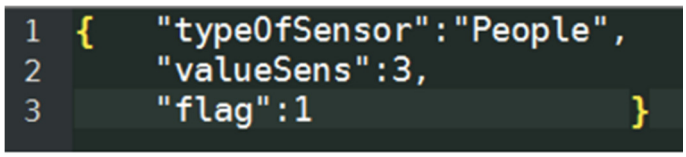

Figura 11: Exemplo de dados salvos em um sensor.
Fonte: Autoria própria.

- Preferências: tem a função de armazenar as preferências definidas pelo usuário em relação aos componentes da casa, por exemplo a temperatura em que um aquecedor deve ser ligado, ou desligado. Tipo de componente, valores para ser ligado e desligado, assim como o valor em que o componente deve ser definido são as características dessa estrutura. A estrutura "Preferências" é demonstrada na Figura 8 (Linhas 18 a 21). A Figura 12 contém um exemplo de preferência relativa a um aparelho de ar condicionado programado para ligar quando o ambiente estiver com 30ºC ou mais, desligar a 20ºC e ser utilizado com sua temperatura em 22ºC.

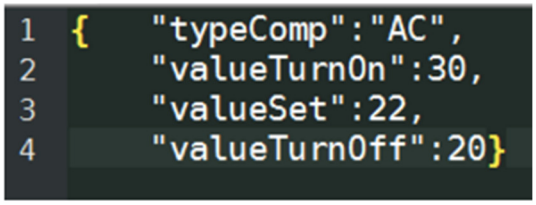

Figura 12: Exemplo de preferência relacionada ao ar condicionado.
Fonte: Autoria própria.

Cada uma das estruturas citadas cria para si um arquivo compartilhado para ser acessado pelas funções e demais objetos do sistema. Esses arquivos ficam salvos no servidor e podem sofrer alterações em suas configurações caso seja necessário, o que vem sendo utilizado para a simulação dos dados dos sensores. Essas alterações podem ser realizadas por programas externos ou pela interface principal do servidor, acessível por meio de autenticação.

4.1.2 Uso do Sistema

Uma vez que o sistema é inicializado, ele publica no endereço local suas principais funções, tornando-as acessíveis dentro da rede local, ou, se for suportado pela estrutura, via conexão externa, permitindo assim acesso pelo endereço IP do servidor ou nome da máquina, se for o caso, autorizando o uso de *webviews* em aplicativos para controle, ou até mesmo o acesso simples por meio de um navegador. Dentro desse sistema, após definido o método de

acesso, como por exemplo o aplicativo para celulares Android que será apresentado na próxima seção, o usuário tem acesso a diversas funções do sistema.

De forma bastante simples e intuitiva, o usuário tem acesso à funções como trancar portas, acender luzes e ligar aparelhos de aquecimento, assim como observar o funcionamento dos mesmos e os dados lidos em seus sensores, conforme ilustrado na Figura 13.

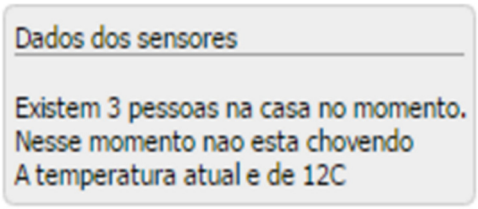

Figura 13: Visualização dos dados dos sensores obtida pelo sistema.
Fonte: Autoria própria.

Quando uma função é utilizada empregando-se desses métodos de entrada, o servidor faz todo o tratamento da informação, e, se for necessário, realiza as mudanças nos objetos solicitados pelo usuário, enquanto ainda mantém sua autonomia nos casos da mudança se fazer necessária, por exemplo para seguir uma configuração de preferência.

O sistema também, além de responder a ordens do usuário, tem a sua parte automatizada para controle sem interferências externas. Esta parte automatizada tem por objetivo um aumento da comodidade e conforto do usuário, uma vez que ele assim terá suas preferências atendidas antes mesmo de precisar manualmente realizar qualquer ação. Um exemplo simples de uma ação automatizada que o sistema atende é a garantia de que quando a temperatura estiver abaixo da mínima definida pelo usuário em sua configuração os aquecedores serão ligados de forma autônoma para o conforto do mesmo.

4.2 APLICATIVO ANDROID

O aplicativo Android foi desenvolvido baseado na premissa de que o mesmo deveria ser um modo de acesso simples às funções do servidor central de controle da casa. Assim sendo, foram adotadas as medidas de utilizar-se de telas de navegação referentes a cada tipo de dispositivo de interesse, e tratar no interior de cada uma dessas telas os dados utilizando-se de *webviews* acessando o servidor. Isto é, o aplicativo requer uma conexão funcional com a *internet*, e que o servidor principal esteja funcionando.

A configuração inicial do aplicativo requer que o usuário entre com o endereço IP do servidor para realizar as conexões, e o opere baseando-se no sistema de navegação apresentado

no diagrama. Ilustrado na Figura 14, o diagrama de navegação do aplicativo *Android* demonstra como cada tela do aplicativo está ligada entre si, permitindo uma visão abstrata de seu funcionamento, que se inicia na tela de autenticação, para em seguida passar pelas diversas outras telas dentro de todo o aplicativo. Essas telas são melhores descritas e mostradas na próxima seção.

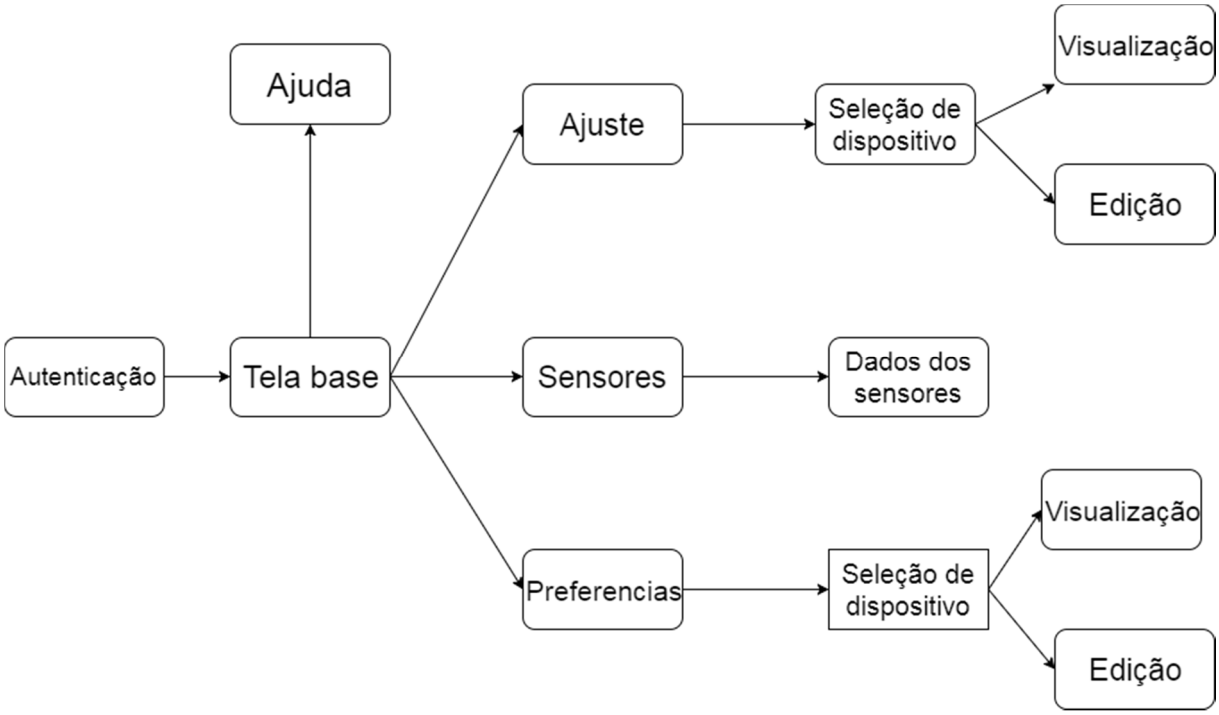

Figura 14: Fluxograma representativo do aplicativo móvel.
Fonte: Autoria própria.

4.2.1 Telas do Aplicativo

Conforme mostrado no diagrama de navegação da Figura 14, o aplicativo, ao ser inicializado no celular, abre em uma tela de autenticação, na qual o usuário deve entrar com o seu nome de usuário e senha. A criação destes deverá ser realizada junto ao administrador do projeto, para que haja maior controle sobre o acesso direto ao uso do aplicativo. O sistema utilizado para a validação e manutenção das contas de usuário é um *framework* aberto ao público fornecido pelo Google, chamado *FireBase Auth*[5], ele testa os dados fornecidos no *login* no aplicativo e os compara com o banco de dados disponível ao administrador.

Foi realizada a escolha de não permitir ao usuário a criação de novas contas pelo aplicativo para assim manter uma quantidade menor de contas ativas originalmente e

[5] Disponível para acesso em: https://firebase.google.com/docs/auth/

dificultando, em um primeiro momento, ações de má fé em um sistema consideravelmente perigoso para tal.

A seguir, a Figura 15 ilustra a tela inicial do aplicativo móvel, a tela de *login*.

Figura 15: Visualização da tela de *login*.
Fonte: Autoria própria.

Após a autenticação ter sido efetuada de forma bem-sucedida no sistema, o usuário é redirecionado para uma tela de abertura com as opções de navegação, conforme mostradas no diagrama, incluindo: sensores, ajustes e preferências, além de ícones direcionando para uma tela de ajuda, ilustrada na Figura 16, e para mudança de idioma do aplicativo, atualmente

disponível para uso em português brasileiro e inglês americano. A Figura 17 demonstra essa tela.

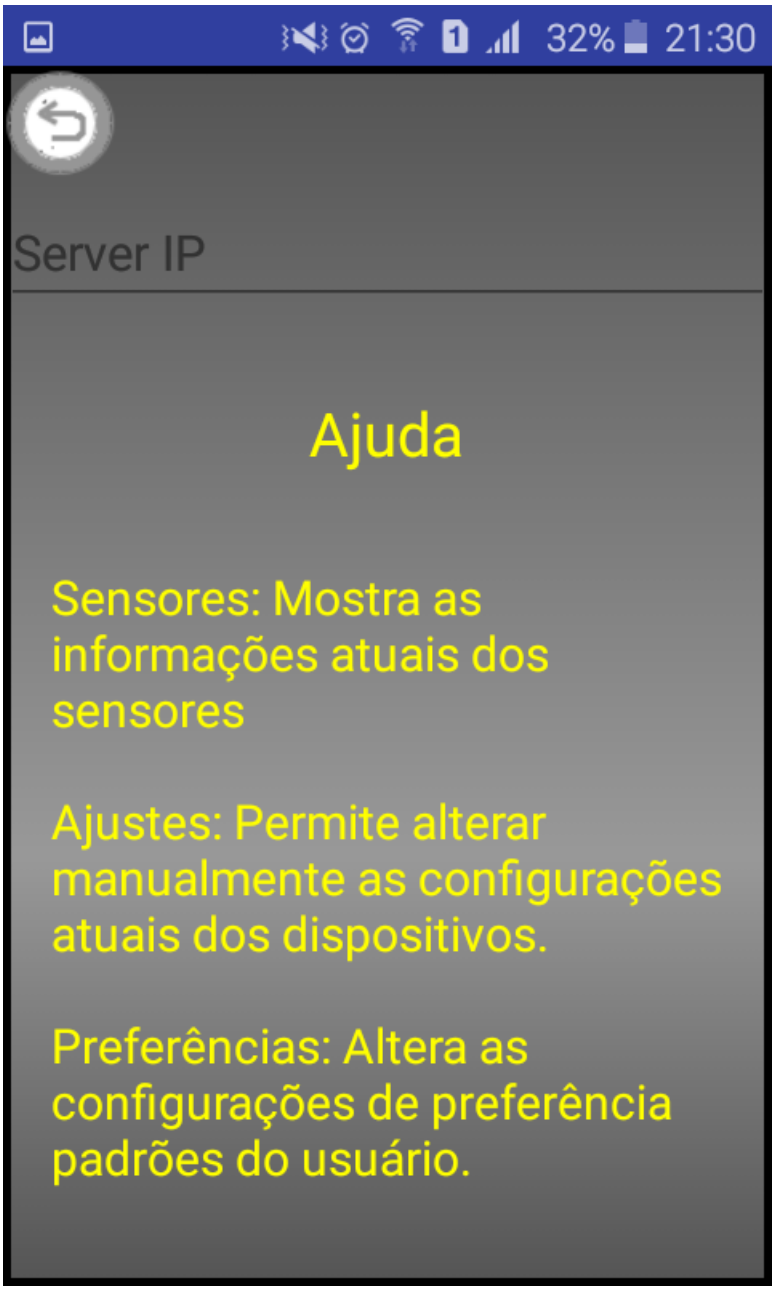

Figura 16: Visualização da ajuda no aplicativo.
Fonte: Autoria própria.

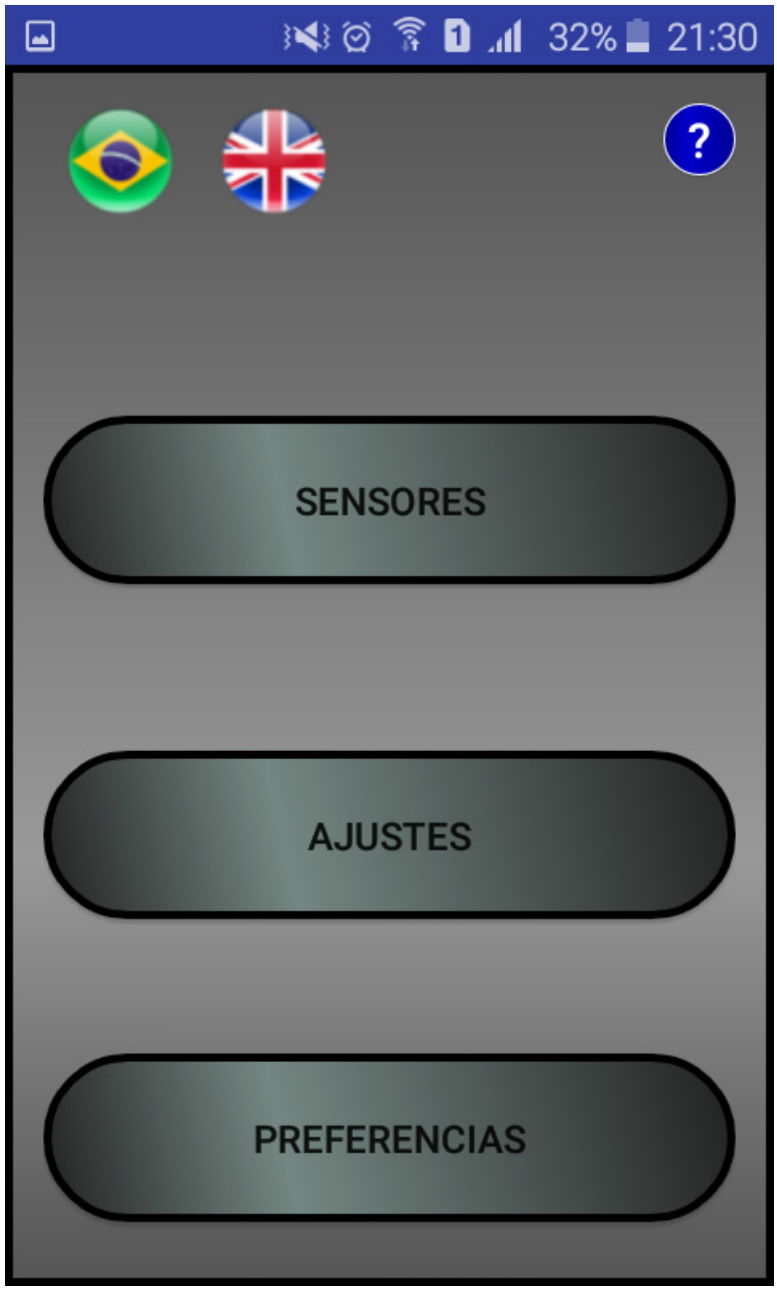

Figura 17: Visualização da tela inicial.
Fonte: Autoria própria.

A opção de sensores tem por função apenas dizer ao usuário os dados dos sensores mais sensíveis a ele, com o intuito de manter a informação aberta e de fácil acesso. Assim, permitindo que o usuário a qualquer momento tenha a ciência do número de pessoas no interior da casa, temperatura e outras informações relevantes para este. Essa tela tem um baixo índice de interação, visto sua função ser de apenas uma visualização dos dados, e não sua manutenção ou alteração, ver figura Figura 18.

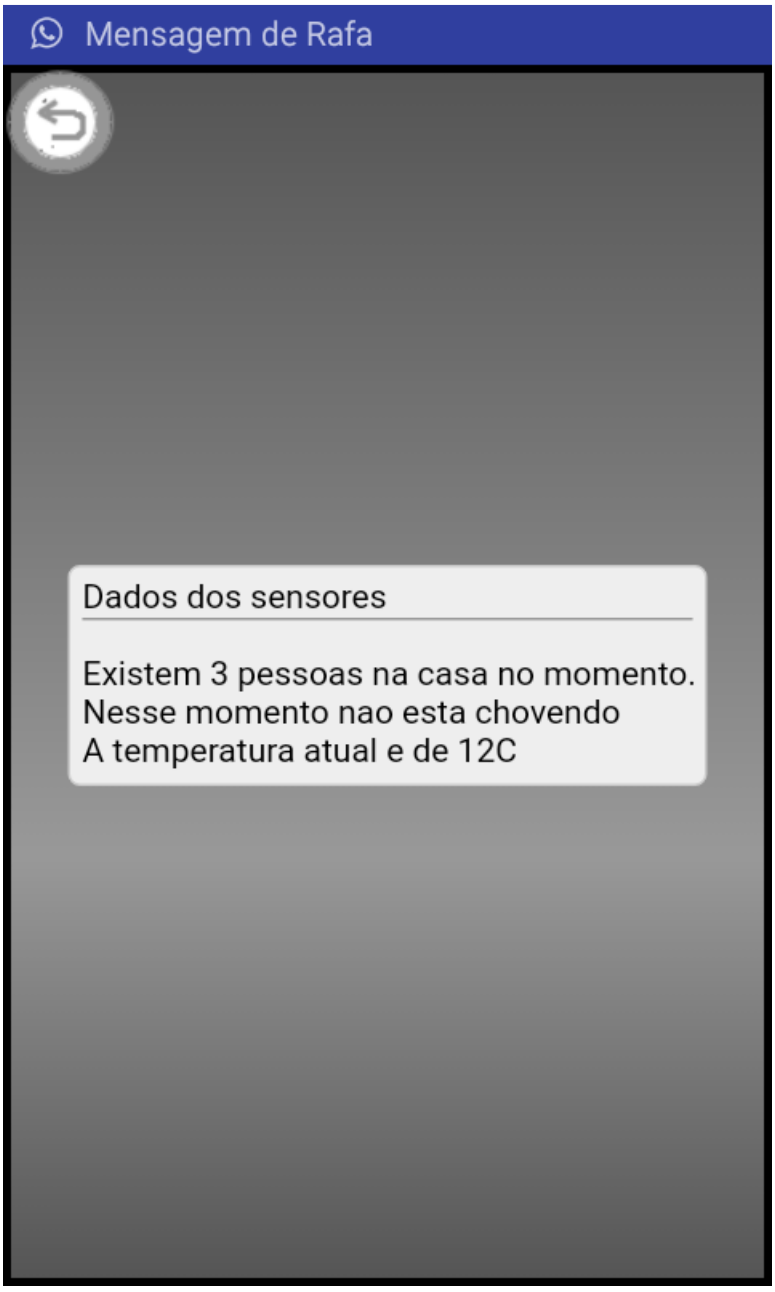

Figura 18: Tela do aplicativo mostrando o *status* dos sensores.
Fonte: Autoria própria.

Na opção de ajuste, o usuário é apresentado a uma vasta lista de opções de dispositivos que ele possa vir a desejar alterar ou conferir os *status* atuais (ver Figura 19). Os dispositivos nessa fase são a contraparte mostrada no aplicativo para os componentes anteriormente explicados no desenvolvimento do sistema. Dentro de cada uma das opções, o aplicativo mostra ao usuário *webviews* remetendo ao servidor, apresentando um passo-a-passo para a alteração de cada dado do sistema (ver Figura 20), assim como a opção de visualizar um mapa pré-definido do ambiente para melhor localização em relação às numerações relativas a cada um

dos dispositivos desejados (ver Figura 21). A Figura 22 ilustra a opção de ajuda dentro de ajustes.

Figura 19: Visualização da tela de ajuste.
Fonte: Autoria própria.

Figura 20: Portas dentro da opção ajustes.
Fonte: Autoria própria.

Ainda dentro da opção de ajustes, há a opção de mexer com portas (ver Figura 20), que apresenta também um menu com um mapa das portas da residência, conforme a Figura 21.

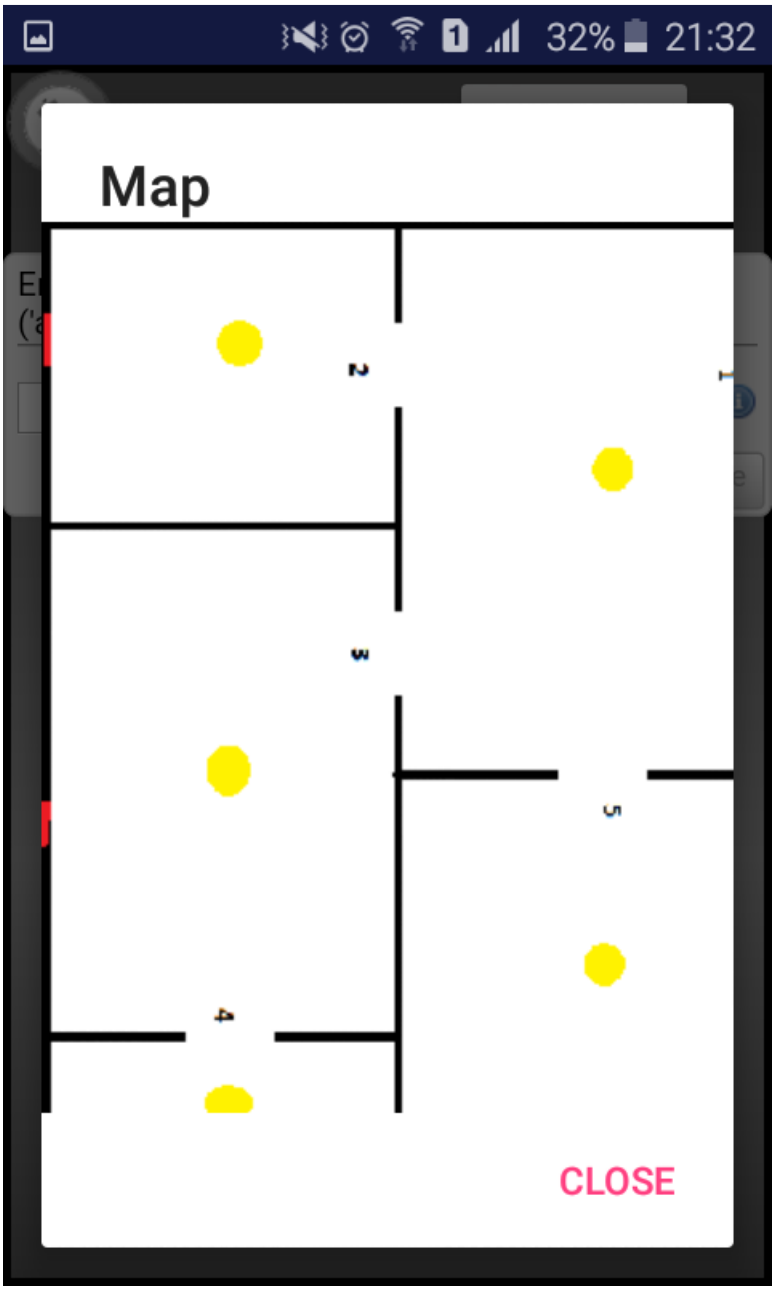

Figura 21: Mapa das portas da residência.
Fonte: Autoria própria.

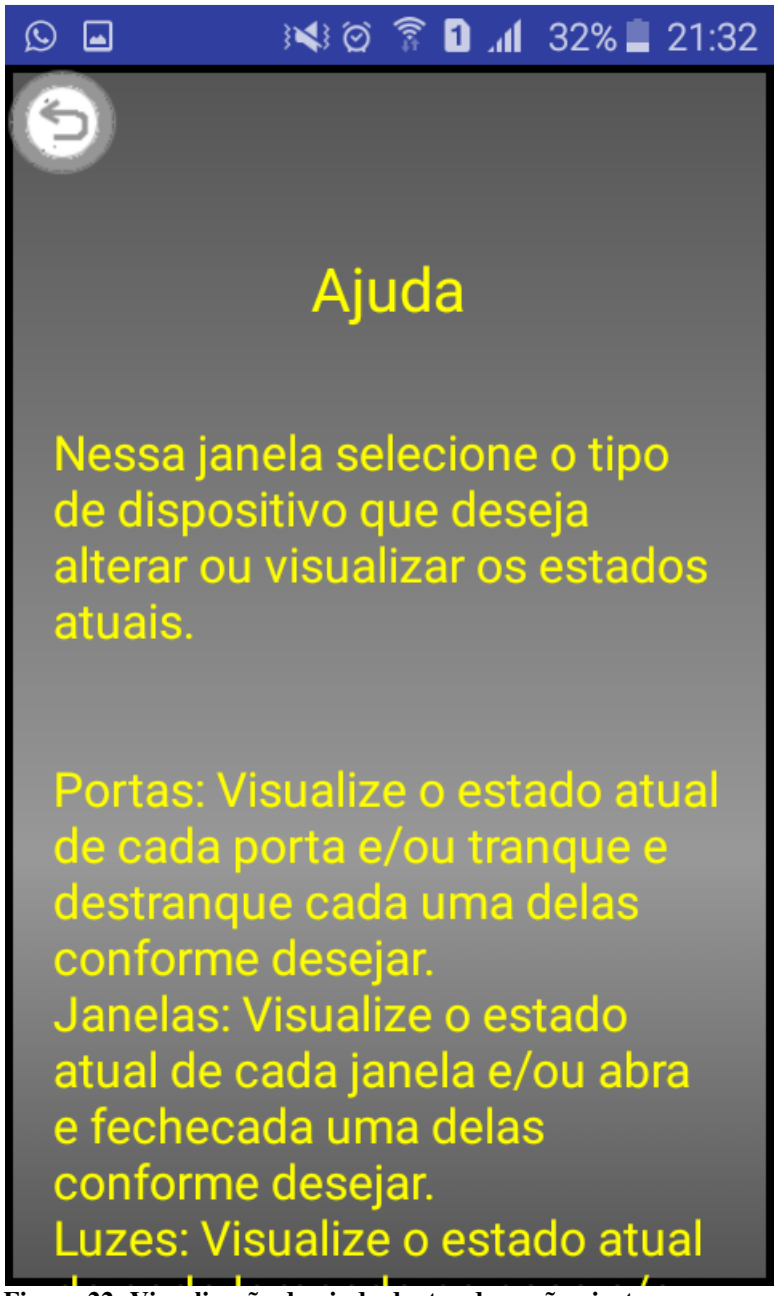

Figura 22: Visualização da ajuda dentro da opção ajustes.
Fonte: Autoria própria.

As figuras 23 e 24 ilustram as opções de ajuste de janelas e luzes, respectivamente.

Figura 23: Ajuste de janelas.

Fonte: Autoria própria.

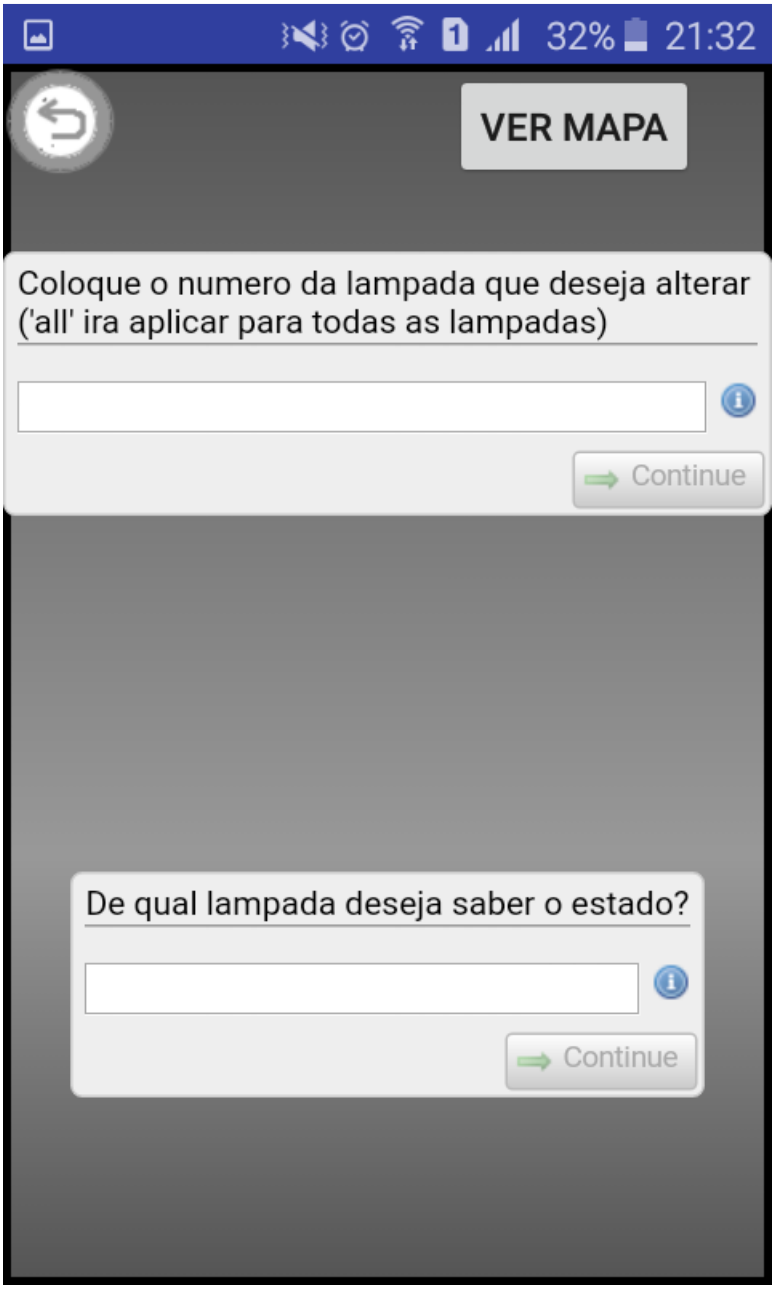

Figura 24: Ajuste de lâmpadas.
Fonte: Autoria própria.

As figuras 25 e 26 apresentam o ajuste do aquecedor e o ajuste do ar condicionado, respectivamente.

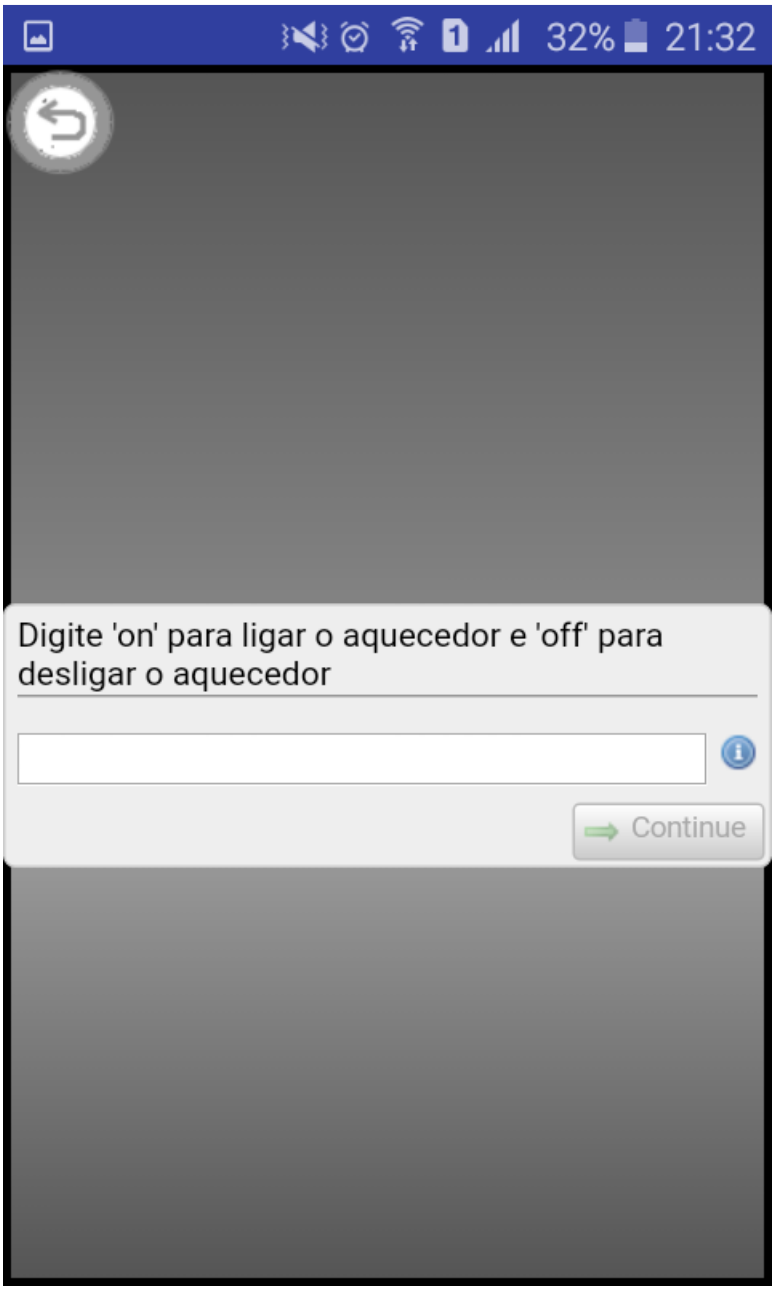

Figura 25: Ajuste do aquecedor.
Fonte: Autoria própria.

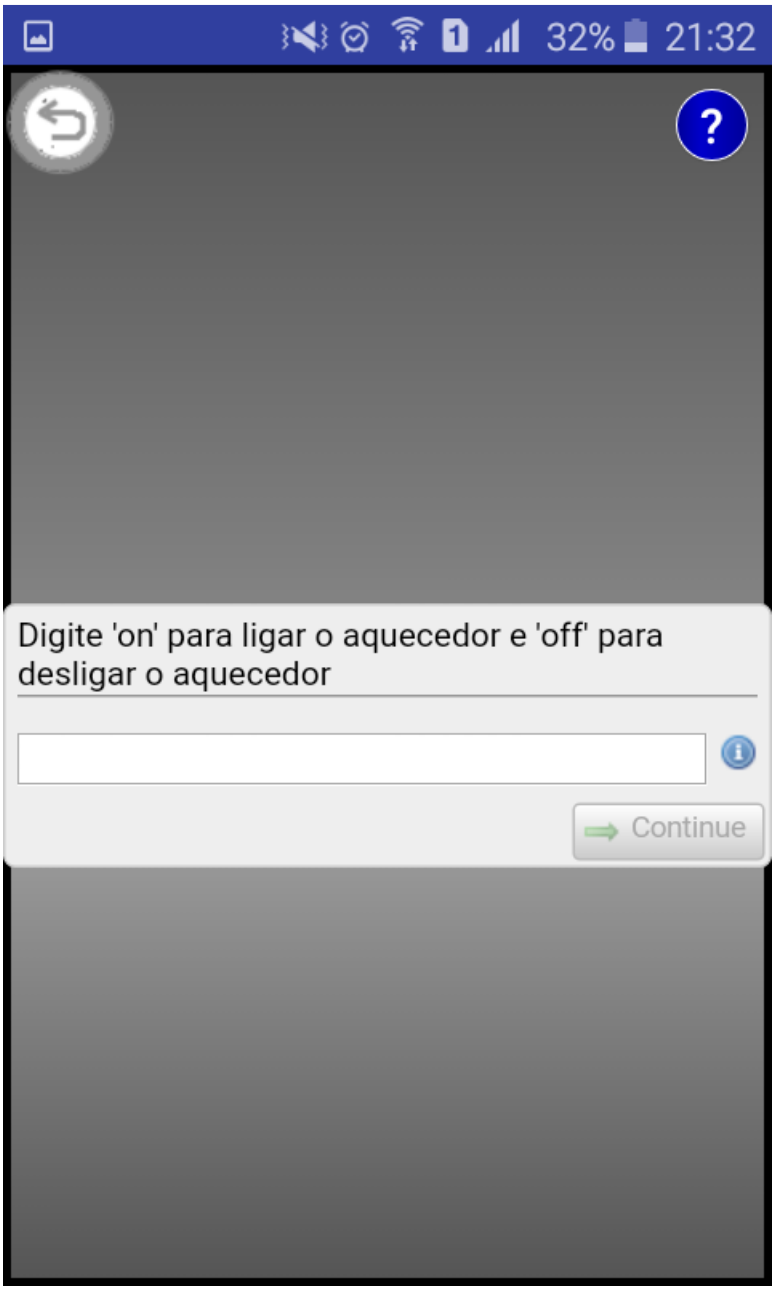

Figura 26: Ajuste do ar condicionado

Fonte: Autoria própria.

A terceira opção dada na tela inicial do aplicativo é a de preferências, que permite a alteração de certas preferências definidas pelo usuário. Essa opção apesar de permitir alteração de preferências mais simples como temperaturas desejadas, não permite a alteração de dados mais sensíveis, que deve ser realizada diretamente no servidor, focando em uma maior segurança e com o intuito da diminuição de erro humano, como por exemplo poderia ocorrer em uma preferência relativa a regras de fechaduras. A Figura 27 demonstra a tela de preferências com as opções disponíveis, as Figuras 28 a 32 ilustram um exemplo de

visualização de preferências do aquecedor e sua alteração e a Figura 33 apresenta a opção de preferências do ar condicionado.

Figura 27: Tela de preferências.

Fonte: Autoria própria.

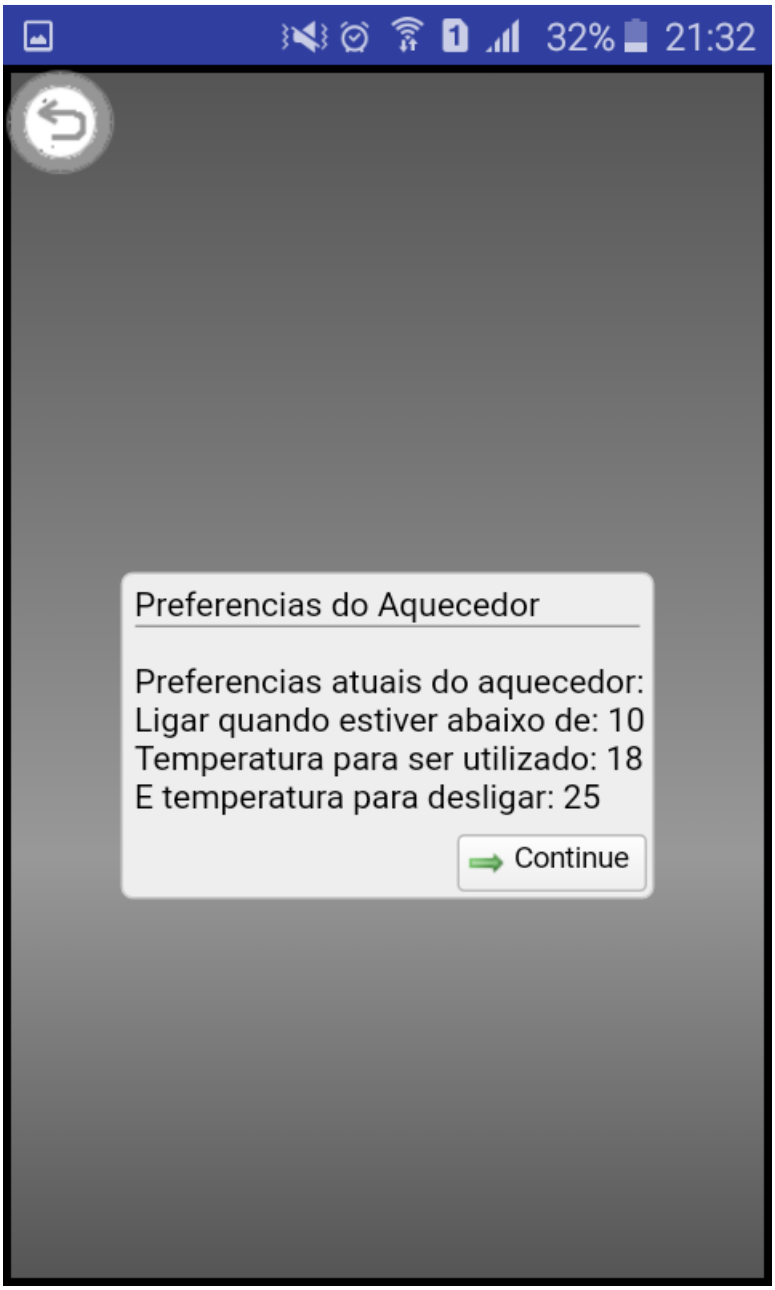

Figura 28: Preferências do aquecedor.
Fonte: Autoria própria.

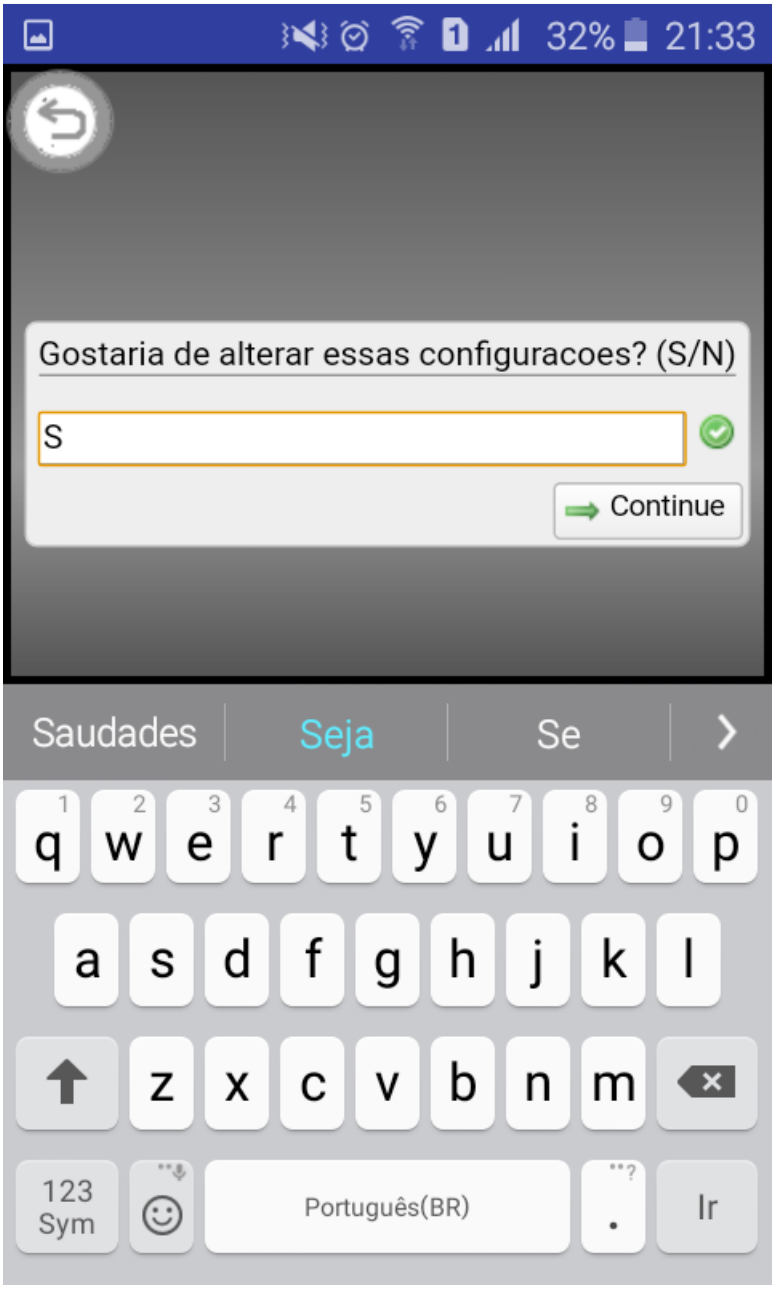

Figura 29: Alteração de configuração do aquecedor.

Fonte: Autoria própria.

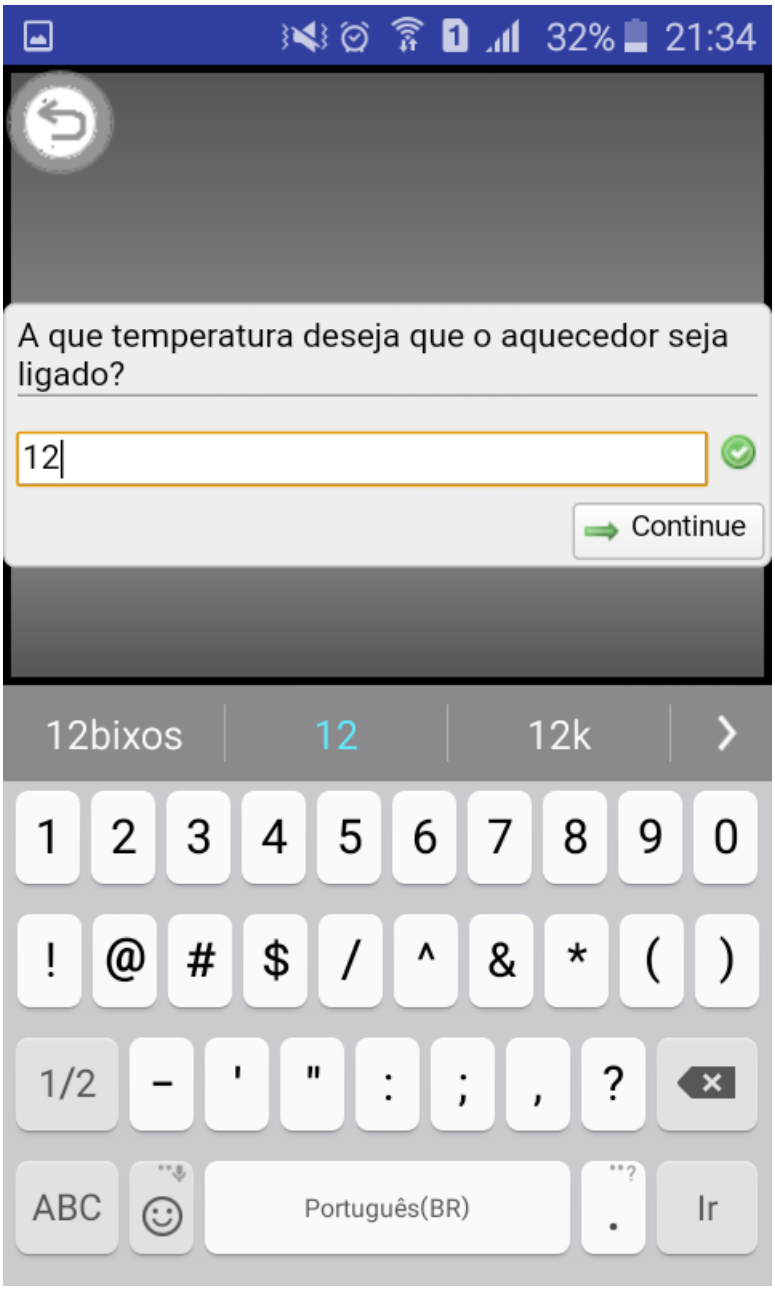

Figura 30: Alteração de temperatura para ligar o aquecedor.
Fonte: Autoria própria.

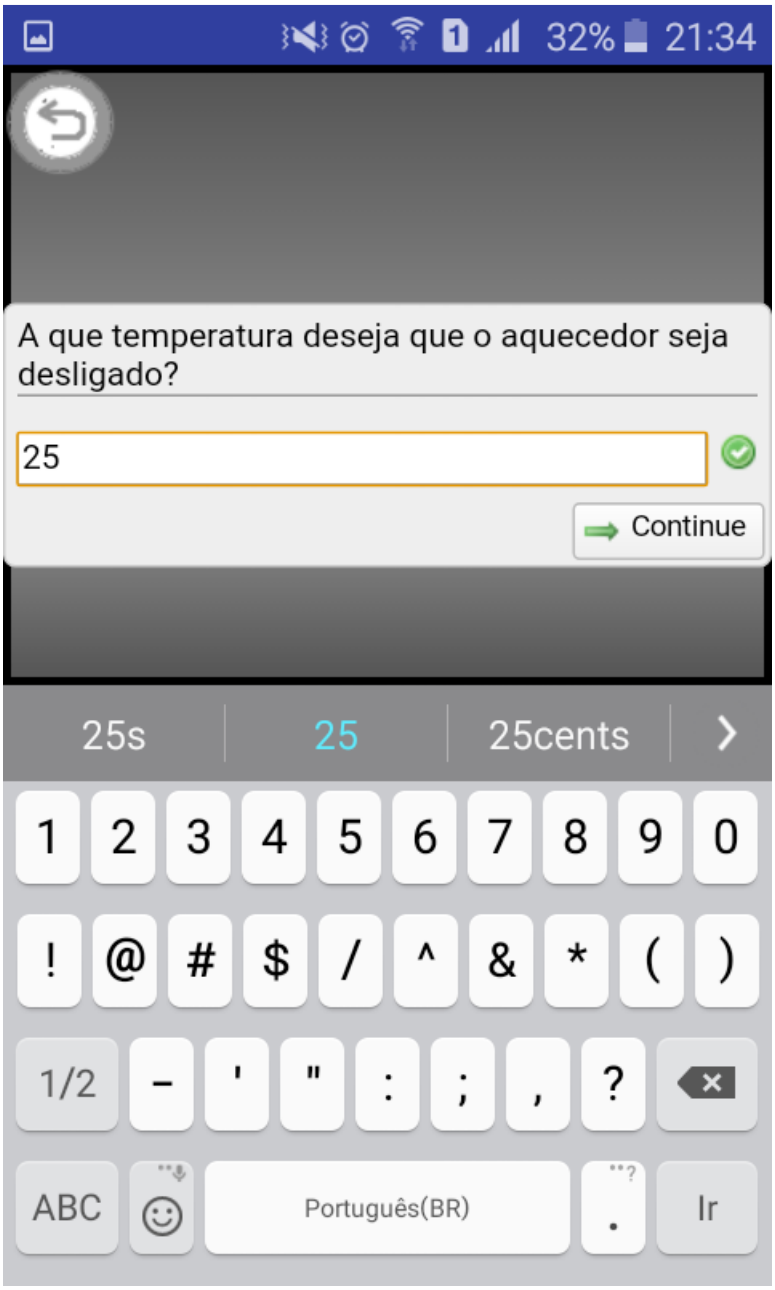

Figura 31: Alteração de temperatura de desligamento do aquecedor.

Fonte: Autoria própria.

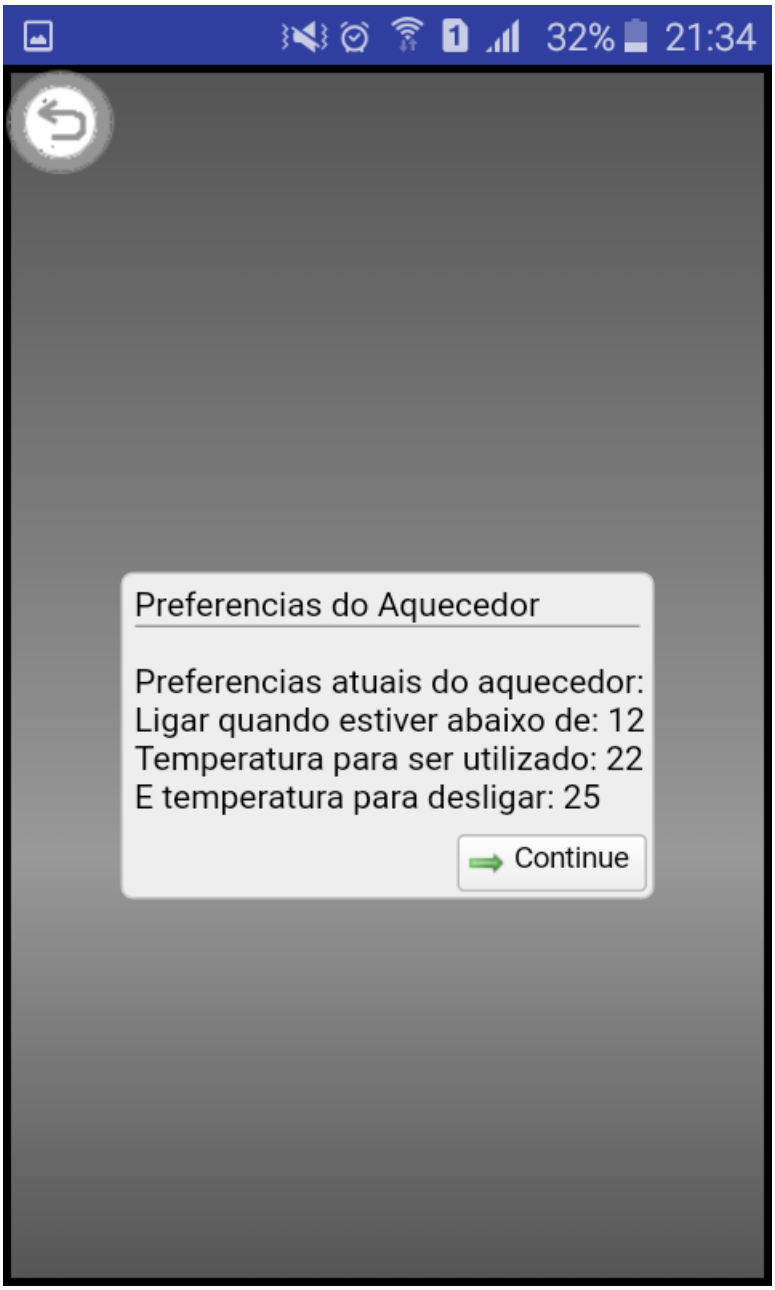

Figura 32: Preferências modificadas do aquecedor.
Fonte: Autoria própria.

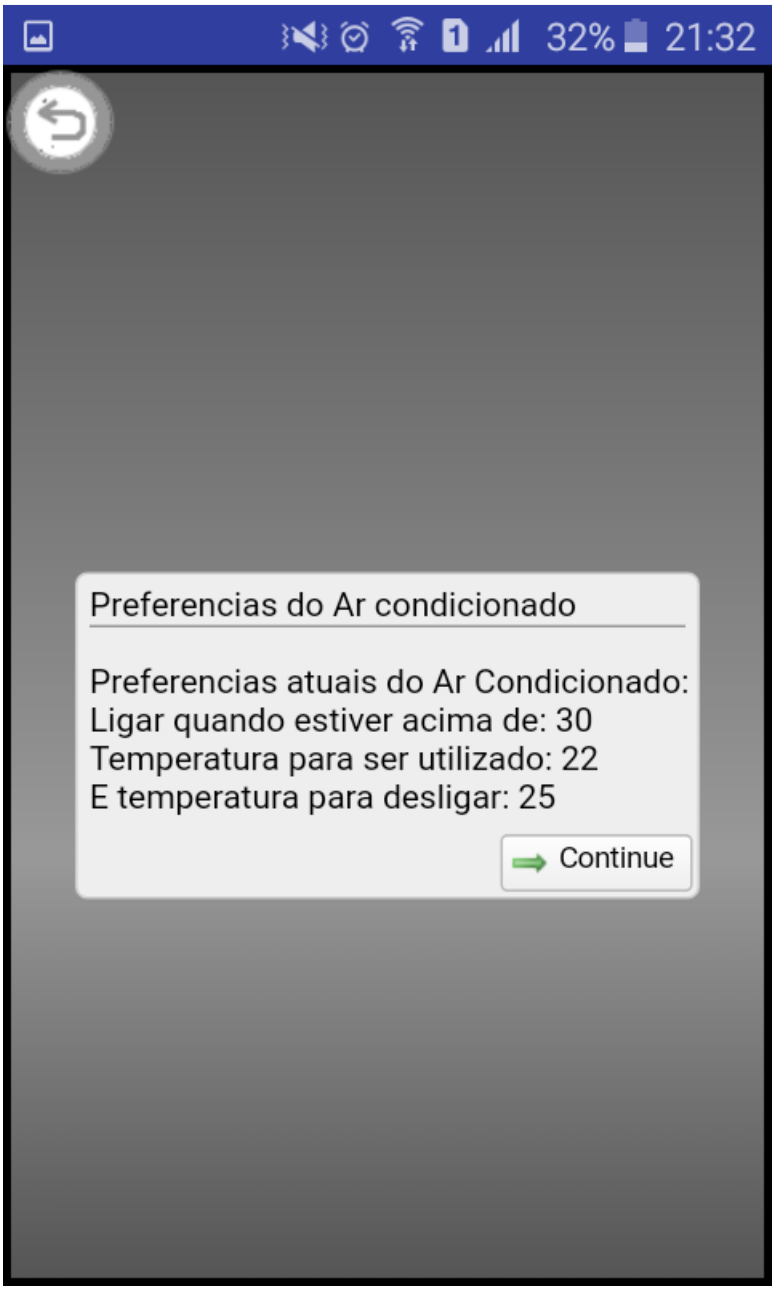

Figura 33: Preferências do ar condicionado.
Fonte: Autoria própria.

5 CONSIDERAÇÕES FINAIS

Em uma sociedade com cada vez mais computadores alterando o modo que as pessoas levam suas vidas, muitas vezes os indivíduos são direcionados a uma vida cada vez mais automatizada e dependente da tecnologia. Existem linhas de raciocínio que tratam isso como algo a ser evitado, outras de que é algo a ser perseguido. Independentemente da linha que esteja certa ou errada, se é que há um certo e errado em questões como essa, ambas convergem no fato de se aceitar que tal movimento é em si irreversível.

Sabendo-se que tais avanços irão ocorrer de uma forma ou de outra, é importante lembrar que por mais que o uso excessivo da tecnologia seja apontado por alguns pesquisadores e médicos como um dos causadores de problemas de saúde e problemas de sono (VOLPI, 2012), pode-se aproveitar dessa mesma tecnologia para auxiliar pessoas, automatizando ambientes, por exemplo o quarto, deixando uma temperatura agradável, configurando a iluminação para que o morador possa dormir confortavelmente. A vida pode ser facilitada e auxiliada pela criação de novas tecnologias em um mundo cada vez mais conectado. Esse projeto se trata de mais um desses avanços sendo colocado a favor do público de uma forma bastante aberta, onde em boa parte é necessária apenas a existência de uma casa e um computador funcional para se iniciar a automatização de uma residência.

Também é possível ver que nos dias atuais boa parte da população tem acesso a um *smartphone*, às vezes nem tendo um aparelho computador doméstico. Por essa razão, fez-se lógica a escolha dessa plataforma para criar o ponto de acesso mais simplificado e menos técnico ao servidor, por mais que ele ainda possa ser acessado apenas pelo computador, mas o uso frequente de celulares tornou a opção de algo feito para *smartphones* mais viável e acertada.

Criando-se uma nova opção tecnológica para facilitar o dia a dia das pessoas, procura-se fazer um serviço para uma sociedade com cada vez menos tempo livre, com cargas de trabalho e responsabilidades cada vez maiores, na contramão do que se espera, permitindo que essas pessoas possam ter um pouco mais de tempo livre, pelo simples fato de não mais precisarem ocupar suas mentes e tempo fazendo atividades que um computador pode realizar com um mínimo de processamento e tempo.

O projeto quando iniciado, parecia se tratar de uma utopia que dificilmente poderia ser alcançada, como visto em vídeos pela internet e demonstrações de preços elevados divulgados

pelas empresas. Mas após realizar as pesquisas e o desenvolvimento percebeu-se que a proposta não era tão inatingível quanto parecia.

O projeto aqui se trata, em sua grande maioria, de uma simulação teórica, uma vez que não pode ser implantada ainda a uma moradia sem que mais algum trabalho seja aplicado. Porém ele comprova que, de forma teórica, o conceito de casas inteligentes pode ser desenvolvido de uma forma a se tornar acessível e constante na vida das pessoas sem a necessidade de investimentos muito altos, ou de tecnologias ainda impossíveis.

Um dos principais obstáculos que foram encontrados durante o desenvolvimento de todo o projeto foram a falta de documentações e artigos científicos tratando tanto do assunto de automatização e desenvolvimento das casas inteligentes quanto da linguagem de programação *Clean* (NÖCKER et al., 1991). Os artigos publicados que foram encontrados no início do projeto eram focados em segurança e em economia de energia, assuntos importantes, mas que não são o foco desse projeto, porém são temas possíveis para trabalhos futuros.

Tratando-se de projetos e desenvolvimentos futuros, no presente documento foi apresentada a teoria e um sistema funcional de controle, mas que em seu futuro poderia, e deveria ter sua demonstração física criada, de modo a poder provar de forma prática a viabilidade de uma casa capaz de tomar conta dessas atividades que auxiliem seus moradores.

REFERÊNCIAS

ALAM, M. R.; REAZ, M. B. I.; ALI, M. A. M. **A Review of Smart Homes - Past, Present, and Future**. 2012. Disponível em: <http://ieeexplore.ieee.org/document/6177682/>. Acesso em: 12 de setembro de 2016.

AMAZON. **Alexa Voice Service**. Disponível em: <https://developer.amazon.com/alexavoiceservice>. Acesso em: 19 de novembro de 2016.

AMAZON. **Amazon Echo - AlexaEnabled**. Disponível em: <https://www.amazon.com/AmazonEchoBluetoothSpeakerwithWiFi-Alexa/dp/B00X4WHP5E>. Acesso em: 18 de novembro de 2016.

APPLE. **IOS - Casa - Apple (BR)**. Disponível em: <https://www.apple.com/br/ios/home/>. Acesso em: 18 de novembro de 2016.

BING, K. et al. **Design of an Internet of Things-based Smart Home System**. 2011. Disponível em: <http://ieeexplore.ieee.org/document/6008384/>. Acesso em: 15 de setembro de 2016.

GAIKWAD, P. P.; GABHANE, J. P.; GOLAIT, S. S. **A Survey based on Smart Homes System Using Internet-of-Things**. 2015. Disponível em: <http://ieeexplore.ieee.org/document/7259486/>. Acesso em: 15 de setembro de 2016.

GOOGLE. **Google Home - Made by Google**. Disponível em: <https://madeby.google.com/home/>. Acesso em: 20 de novembro de 2016.

KAVITHA, V. **Context Aware Approach for Smart Homes**. 2009. Disponível em: <http://dl.acm.org/citation.cfm?id=1670380>. Acesso em: 2 de abril de 2017.

KOOPMAN, P. et al. **Functional Programming in Clean - Draft**. 2002. Disponível em: <ftp://ftp.cs.kun.nl/pub/Clean/papers/cleanbook/CleanBookI.pdf>. Acesso em: 19 de novembro de 2016.

NÖCKER, E. et al. **Concurrent Clean**. 1991. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.36.2676&rep=rep1&type=pdf>. Acesso em: 18 de novembro de 2016.

PLASMEIJER, R.; ACHTEN, P.; KOOPMAN, P. **An Introduction to iTasks: Defining Interactive Work Flows for the Web**. 2008. Disponível em: <http://www.mbsd.cs.ru.nl/publications/papers/2008/plar08-iTasks_CEFP2007_Revised.pdf>. Acesso em: 17 de novembro de 2016.

PLASMEIJER, R.; EEKELEN, M. van. **Concurrent Clean Language Report (version 2.0)**. 2001. Disponível em: <http://www.cs.ru.nl/ clean/>. Acesso em: 18 de novembro de 2016.

ROSE, B. **Home Networks: A Standards Perspective**. 2001. Disponível em: <http://ieeexplore.ieee.org/document/968816/>. Acesso em: 4 de março de 2017.

SAMSUNG. **Smartthings. Add a little smartness to your things**. Disponível em:

<https://www.smartthings.com/>. Acesso em: 17 de maio de 2017.

SIMS, G. **I want to develop Android Apps - What languages should I learn?** 2014. Disponível em: <http://www.androidauthority.com/wantdevelopandroidappslanguageslearn391008>. Acesso em: 13 de março de 2017.

VOLPI, D. **Heavy Technology Use Linked to Fatigue, Stress and Depression in Young Adults**. 2012. Disponível em: <http://www.hufingtonpost.com/davidvolpimdpcfacs/Technologydepression_b_1723625.html>. Acesso em: 18 de maio de 2017.

WANG, W.-Y. et al. **A Context-Aware System for Smart Home Applications**. 2005. Disponível em: <http://dl.acm.org/citation.cfm?id=2113617>. Acesso em: 4 de março de 2017.

WINK. **Wink | A Simpler, Smarter Home**. Disponível em: <https://www.wink.com/>. Acesso em: 17 de maio de 2017.

ZHANG, Y. et al. **A Solution for Low Cost and High Performance Smart Home Networking**. 2011. Disponível em: <http://ieeexplore.ieee.org/document/6218521/>. Acesso em: 4 de março de 2017.

ÖZKAN, H. A.; AYBAR, A. **A Smart Air Conditioner in Smart Home**. 2011. Disponível em: <http://ieeexplore.ieee.org/document/7555599/>. Acesso em: 18 de setembro de 2016.

APÊNDICE A – DOCUMENTAÇÃO CLEAN

A.1 DEFINIÇÃO

Clean é uma linguagem de programação puramente funcional com o intuito de desenvolvimento de aplicações capazes de simular o mundo externo. Sendo uma linguagem funcional, e como a maioria destas, *Clean* é implementado baseado em técnicas de reescrita de grafos.

Um dos fatores de maior importância para o uso prático de *Clean* é o seu sistema de singularidade de tipos o que permite ao programador a criação de objetos com o único intuito de trabalharem em um *framework* puramente funcional e assim criar interfaces com ligações diretas com o mundo externo ao aplicativo em si.

A.2 SINTAXE DA LINGUAGEM

A sintaxe utilizada na linguagem *Clean* é bastante similar com a de boa parte das linguagens funcionais mais modernas, mas com pequenas singularidades sintáticas. Quando uma função é definida, sua aridade e tipos também são definidas juntamente. Conforme o modelo da Figura 34A, onde no cabeçalho da função é definido o tipo e quantos dados devem ser fornecidos (Linha 1), e nas linhas posteriores a programação do que deve ser feito com tais dados e por fim o retorno para o usuário (Linha 2)

```
1 NomeDaFunção :: Tipo dos Dados de Entrada -> Tipo dos Dados de Saída
2 NomeDaFunção dadoRecebido = dadoASerRetornado ao usuário
```

Figura 34A: Modelo para declaração de função em Clean.
Fonte: Autoria própria.

A.3 TIPOS DE DADOS E FUNÇÕES

Os tipos de dados básicos pré-definidos pela linguagem são:

- Int - Se referindo a numerais inteiros de até 32 *bits*;
- Real - Para o uso de números reais com casas decimais, até 64 *bits;*
- Char - Para algarismos ASCII de 8 bits;
- Bool - Para armazenar valores *booleanos* alternando entre verdadeiro e falso.

Na Figura 35A tem-se um exemplo de função de soma criada utilizando os tipos de dados. Neste caso a função recebe dois números inteiros e retorna um terceiro, representando a soma dos mesmos. Já em seguida, na Figura 36A é possível ver um exemplo com a entrada de um número inteiro, e os dados da saída em um *boolean* (Linha 1), sendo este verdadeiro caso o número seja par, checando nesse caso o fato baseado no resto da divisão do número desejado por 2 (Linha 3), e falso em caso de um número ímpar (Linha 4).

```
1 Soma :: Int Int -> Int
2 Soma x y = x+y
```

Figura 35A: Exemplo de função de soma.
Fonte: Autoria própria.

```
1 EhPar :: Int -> Bool
2 EhPar x
3 | x rem 2 == 0 = True
4 = False
```

Figura 36A: Exemplo de função para definir se um número é par.
Fonte: Autoria própria.

A linguagem também possui por padrão em sua implementação algumas estruturas de dados já definidas, como listas, tuplas, *arrays* e *strings* sendo função do programador apenas denotar por meio de símbolos a estrutura desejada.

- [1, 2 ,3] - O uso de colchetes em torno de símbolos separados por vírgula define a criação de uma lista;
- (1, ‘a’) - Tuplas são definidas por seus componentes separados por vírgula no interior de um par de parênteses;
- {1, 2, 3} - Um *array* é uma estrutura representada pelos elementos inseridos entre chaves;
- “Exemplo” - Uma *String* se trata de um caso especial de *array*, onde esse é demonstrado de uma forma diferente, sendo necessária apenas a sequência de algarismos dentro de aspas, implicando assim o *array.*

```
1 SomaLista :: [Int] -> Int
2 SomaLista [] = 0
3 SomaLista [h:t] = h + SomaLista t
```

Figura 37A: Exemplo de função de soma utilizando uma lista.
Fonte: Autoria própria.

Na Figura 37A tem-se um exemplo de função que utiliza a estrutura de uma lista para realizar a soma de todos os seus elementos e retornar para o usuário esse valor. O algoritmo funciona recebendo uma lista de números inteiros (Linha 1), e caso a lista esteja vazia, ele

retorna o valor 0, denotando o fim de uma recursão e também o valor da soma dos zero elementos ali presentes (Linha 2). No caso da lista não se encontrar vazia, o algoritmo soma o valor do primeiro item da lista ao valor obtido pela função executando novamente com a lista menos o seu primeiro valor (Linha 3).

Funções tem também a capacidade de se referirem umas às outras em caso de ser desejado ao sistema, trabalhando de forma a seguir as ordens em uma ordem cronológica dentro do programa, compilando a medida que se fazem necessárias as linhas (*Lazy Compilation*). Um exemplo de funções se referindo umas às outras pode ser visto na Figura 38A. Onde a função "NaoPar", ao receber um número inteiro, analisa se o mesmo não é par comparando com a função 'EhPar', e caso essa seja falsa retorna um valor verdadeiro para a função 'NaoPar' (Linha 3), em outro caso retorna Falso (Linha 4)

```
1 NaoPar :: Int -> Bool
2 NaoPar x
3 | not (EhPar x) = True
4 = False
```

Figura 38A: Exemplo de função com chamada para outra função.
Fonte: Autoria própria.

Exemplos também de recursão podem ser vistos no caso da função "SomaLista" da Figura 37A, na qual a mesma função é chamada ao final dela, iniciando um ciclo de recursão. Outro exemplo clássico utilizado para demonstrar recursão pode ser visto na Figura 39A com a apresentação de um algoritmo em *Clean* representando *Fibonacci*. Após receber o valor desejado para a posição na sequência de *Fibonacci*, a tarefa passa pelas duas regras de parada do algoritmo (Linhas 2 e 3) quando os valores forem 1 ou 2. Em seguida, inicia as chamadas caso o número desejado esteja acima de zero, realizando chamadas pela própria tarefa com os valores diminuindo a cada recursão, até parar em uma regra e retornar o valor integral.

```
1 Fibo :: Int -> Int
2 Fibo 1 = 1
3 Fibo 2 = 1
4 Fibo x
5 | x>0 = Fibo (x-1) + Fibo (x-2)
6 = 0
```

Figura 39A: Algoritmo para Sequência de Fibonacci em Clean.
Fonte: Autoria própria.

A.4 CRIAÇÃO DE NOVOS TIPOS DE DADOS

A linguagem *Clean* permite, ainda, a criação de novos tipos de dados, fazendo uma alegoria à criação de objetos em linguagens orientadas aos mesmos. Esses novos tipos são também utilizados por funções e tem sua utilidade bem próxima àquela que os tipos básicos possuem, porém com a vantagem de poderem manter várias informações diferentes, e retratar objetos do mundo real com melhor precisão.

```
1  :: Produto =    { preco :: Real,
2                    nome :: String,
3                    estoque :: Int}
```

Figura 40A: Exemplo de novo tipo sendo criado.
Fonte: Autoria própria.

Na Figura 40A tem-se a apresentação da sintaxe de criação de um novo tipo de dado, nesse caso representando como seria demonstrado em um sistema um produto de um supermercado por exemplo, assim listando seu preço (em Real), nome (em *String*) e quantidade em estoque (em inteiro), respectivamente. Após um novo tipo ter sido criado ele passa a estar disponível para ser utilizado por funções assim como se utilizariam os tipos básicos de dados que foram apresentados na Seção A.3.

A.5 ARMAZENAMENTO DE DADOS

Após a criação de um novo tipo de dado há a possibilidade de o sistema, com a utilização das bibliotecas referentes ao *iTasks*, permitir o armazenamento das informações desses dados de uma forma externalizada, de forma que todo o sistema possa ter acesso a todos os objetos para realizar suas tarefas naquela estrutura de dados com maior facilidade. Um exemplo, utilizando a estrutura “Produto”, pode ser visto na Figura 41A, e em seguida, um exemplo de dado criado no arquivo (ver Figura 42A).

```
1  listaProdutos :: Shared[Produto]
2  listaProdutos = sharedStore "Produtos" []
```

Figura 41A: Criação de estrutura para armazenamento de dados.
Fonte: Autoria própria.

```
1 20170611-234653[{"preco":10,"nome":"NomeDoProduto","estoque":5}]
```

Figura 42A: Exemplo de dado criado no arquivo.
Fonte: Autoria própria.

Utilizando o arquivo criado e seus dados armazenados, se torna possível criar funções um pouco mais complexas para a alteração desses dados, como adicionar ou remover um produto (ver Figuras 43A e 44A). Na Figura 43A tem-se uma função sem retorno para o usuário, o que a permite não seguir a sintaxe padrão no cabeçalho da mesma (Linha 1), que obtém os dados do arquivo armazenado e os salva em uma variável para uso a seguir (Linhas 2 e 3). A seguir é adicionado o novo item desejado na lista de produtos. Posteriormente, no exemplo seguinte (ver Figura 44A) vê-se as variáveis a serem recebidas pela função, uma lista e uma identificação (Linha 1), para a seguir ela desempenhar uma recursão testando se algum produto da lista tem o nome desejado, caso o encontre a função remove-o da lista (Linhas 2 e 3).

```
1 addProduto produto =
2     get listaProdutos >>=
3     \prodLis ->
4     (set (prodLis ++ [{preco = produto.preco, nome = produto.nome,
5     estoque = produto.estoque}]) listaProdutos)
```

Figura 43A: Exemplo de função para adicionar um produto na lista.
Fonte: Autoria própria.

```
1 remProduto [x:xs] id
2 |(x.nome == id) = xs
3 | otherwise = [x:remProduto xs id]
```

Figura 44A: Função para remoção de um produto da lista, buscando por seu nome.
Fonte: Autoria própria.

A.6 UTILIZAÇÃO DAS FUNÇÕES POR PARTE DO USUÁRIO

Uma vez com o modelo definido, o próximo passo de interesse é o de divulgação dessas tarefas para o usuário, de modo a permitir que este as execute e possa utilizar de uma forma satisfatória do sistema criado. A linguagem *Clean*, com a adição do *iTasks*, permite que facilmente seja publicado um servidor no computador executando-o localmente, permitindo assim o acesso via navegador por parte do usuário. Essa navegação pode ser realizada diretamente pelo endereço (*link*) da função, ou a partir de um *workflow* com todas as funções disponíveis (ver Figura 45A).

```
1  basicWorkflow :: [Workflow]
2  basicWorkflow = [workflow "Adicionar produto" "Adicione produtos" adicionar]
3
4  Start :: *World -> *World
5  Start world = startEngine [publish "/addP" (WebApp []) (\_-> adicionar),
6  publish "/" (WebApp []) (\_-> browseTasks basicWorkflow)] world
7
8  where
9      browseTasks x = forever (
10             manageWorklist x)
```

Figura 45A: Funções para divulgação das funções para o usuário.
Fonte: Autoria própria.

Para apresentar uma função para utilização pelo usuário externo, faz-se necessário o conhecimento de algumas funcionalidades da linguagem, primariamente sobre aquelas que mostram informação ao usuário e aquelas que obtém dados do mesmo. Essas funções são:

- *ViewInformation* - Usada para mostrar informações por escrito para o usuário, como os dados de um produto, a temperatura, ou mesmo uma mensagem antes do fechamento. Sua sintaxe é dada como a presente na Figura 46A;
- *EnterInformation* - Função para obter dados do usuário, para preenchimento de um formulário ou para obter os dados relativos as variáveis de um objeto. A sintaxe está presente na Figura 47A.

```
1  viewInformation "Texto do Titulo" [] "Texto da janela"
```

Figura 46A: Sintaxe da função *viewInformation*.
Fonte: Autoria própria.

```
1  enterInformation "Texto para o usuário ler" []
2      >>= \variavelParaSalvar
```

Figura 47A: Sintaxe da função *enterInformation*.
Fonte: Autoria Própria.

Com o conhecimento dessas funções, já se torna possível a elaboração de tarefas em cooperação com o usuário, de modo a obter um programa com funcionalidades e interações necessárias para a sua usabilidade. A Figura 48A contém um exemplo de função já preparada para a comunicação com o usuário, uma vez que ela já se encontra disponível e publicada como visto anteriormente nesse capítulo. A função se inicia mostrando ao usuário uma mensagem de texto (Linha 1), para a seguir pedir ao mesmo pelas informações relacionadas (Linhas 2, 3 e 4) para enfim fazer uma chamada ao sistema a para realizar a ação necessária (Linha 5), antes de chamar a função uma outra vez de forma a voltar ao seu início para reutilização (Linha 6).

```
adicionar = viewInformation "Bem-vindo" [] "Adicione novos produtos" >>|
            enterInformation "Qual o nome do produto a ser adicionado?" []
            >>= \nomeP -> enterInformation "Qual o preco do produto?" []
            >>= \precoP -> enterInformation "Qual a quantidade do produto?" []
            >>= \quantP ->addProduto {preco = precoP, nome = nomeP, estoque = quantP}
            -|| adicionar
```

Figura 48A: Exemplo de função pronta para interação do usuário.
Fonte: Autoria própria.