DAS Departamento de Automação e Sistemas
CTC Centro Tecnológico
UFSC Universidade Federal de Santa Catarina

Desenvolvimento de Aplicação para Controle de Dispositivos via Rádio Frequência

Relatório submetido à Universidade Federal de Santa Catarina

como requisito para a aprovação na disciplina

DAS 5511: Projeto de Fim de Curso

Antonio Adalberto Duarte Júnior

Florianópolis, setembro de 2016

Desenvolvimento de Aplicação para Controle de Dispositivos via Rádio Frequência

Antonio Adalberto Duarte Júnior

Esta monografia foi julgada no contexto da disciplina
DAS5511: Projeto de Fim de Curso
e aprovada na sua forma final pelo
Curso de Engenharia de Controle e Automação

Prof. *Eduardo Augusto Bezerra*

_____
Assinatura do Orientador

Banca Examinadora:

Alexandre Cordeiro
Orientador na Empresa

Prof. Eduardo Augusto Bezerra
Orientador no Curso

Pedro Henrique Peralta
Orientador na Empresa

Prof. Rodrigo Castelan Carlson
Avaliador

Tiago Nunes Resmini
Rodrigo Holz Schuler
Debatedores

Agradecimentos

Dedico estas palavras para agradecer as pessoas que fizeram parte de mais essa etapa de minha vida. Especialmente a minha família que ao longo desta trajetória incentivou, patrocinou e testemunhou as duras ausências necessárias para o alcance destes objetivos.

Agradecimento especial ao meu amigo Guilherme Gonçalves, Guiga, por despertar o meu interesse por estudar engenharia na UFSC, além de compartilhar o mesmo teto por muitos anos ao longo do curso.

Agradeço ao Pedro Henrique Peralta pela recomendação à empresa e auxílio ao longo do trabalho, e a todos os membros da Cheesecake Labs, que proporcionaram a oportunidade de realizar este trabalho em um ambiente amigável e de muito crescimento.

A todos os professores da UFSC, colegas e amigos que impulsionaram o meu crescimento ao longo dos anos na universidade. Em especial à turma 2010.1, que certamente foi a maior conquista de minha vida acadêmica.

Por último, mas não menos importante, aos orientadores que sempre estavam presentes para auxiliar, ensinar e resolver problemas. Meu agradecimento em especial ao professor Eduardo Augusto Bezerra, pela orientação no PFC e estágio obrigatório; ao Alexandre Cordeiro pela liderança e orientação do primeiro ao último dia de projeto e aos integrantes da empresa Cheesecake Labs que auxiliaram na execução desse trabalho: Bruno Muller, Eduardo Cardoso, Guilherme Hayashi e Victor Oliveira.

Resumo

O conceito de automação residencial tem mudado ao longo dos anos, incorporando uma gama de possibilidades práticas e econômicas que utilizam a automação, desde a básica até a mais abrangente, em sistemas de integração para diversos ambientes. O objetivo é criar um ambiente prático, confortável, valorizado e seguro, de acordo com os interesses dos usuários. Os dispositivos móveis também foram incorporados como soluções de automação residencial, na maioria das vezes substituindo os diversos controles remotos.

A Cheesecake Labs, empresa situada em Florianópolis, SC, foi contratada para o desenvolvimento de um software que tem como objetivo comunicar um aplicativo móvel com uma placa que envia sinais de rádio frequência para controlar dispositivos comuns encontrados em residências, principalmente nos Estados Unidos, como ventiladores de teto ou portões de garagens. Este projeto, denominado Bond, deve habilitar o usuário a configurar comandos de diferentes controles, permitindo que o morador substitua todos os controles remotos da casa — incluindo aqueles que emitem sinais de rádio frequência — por um aplicativo móvel.

Primeiramente foram feitas reuniões de escopo com a empresa contratante Altus-PCB, de Nova Jérsei – EUA. Em seguida, iniciaram as etapas de design do aplicativo na plataforma iOS/iPad, da Apple Inc. Com posse das telas do aplicativo, iniciaram os esforços para o desenvolvimento do software do aplicativo móvel, desenvolvimento do sistema de back-end em sistema embarcado, descoberta automática de dispositivos via Bonjour, comunicação via REST API entre aplicativo e back-end hospedado em sistema embarcado e, por fim, comunicação serial entre sistema embarcado e placa de rádio frequência.

É importante destacar que a incorporação de controles via rádio frequência e a possibilidade de programação desses pelo próprio usuário, representa uma inovação para os sistemas atuais de automação residencial.

Abstract

The concept of home automation has been changing through the years, adding a variety of practical and economics benefits, in different levels, allowed by the home device's integrations. The goal is to offer comfortable, intelligent, safe and valuable environment according to people desires. The mobile devices are also part of the new era of home automation and are introduced, most of the time, as an alternative to the standard remote controls.

The company Cheesecake Labs, from Florianopolis, SC, was contracted to develop the software in order to communicate a mobile application with a radio frequency board that sends signals to control home devices, like ceiling fans and garage doors. The project, called Bond, aims to allow the user to configure different commands, enabling people to replace their common home controls with the Bond mobile application and radio frequency board.

Firstly, scope definition meetings were done to better understand the client's needs. Secondly, the design of the mobile application was developed. Once the mock-ups were completed, the development of the mobile application was done, followed by the back-end hosted on the embedded system, automatic discovery of devices through Bonjour, REST API communication between mobile application and embedded system, and, finally, the serial communication between the embedded system and radio frequency board.

It's important to mention that adding radio frequency controlled devices to the home automation scope represents an advance on the current home automation technologies, since most of the controlled devices works with infrared or Wi-Fi signals.

Lista de Ilustrações

Sumário

Simbologia

API – Application Programming Interface

HTML – Hyper Text Markup Language

CSS – Cascading Style Sheets

JSON – JavaScript Object Notation

HTTP – Hypertext Transfer Protocol

XML – Extensible Markup Language

REST – Representational State Transfer

IP – Internet Protocol

MVC – Model View Controller

MVP – Minimum Viable Product

MCU – Microcontroller Unit

EMS – Electronic Manufacturing Services

OEM – Original Equipment Manufacturers

UI – User Interface

1 Introdução

1.1 Contextualização

Com os avanços das tecnologias de automação residencial, através de investimentos de renomadas empresas [1] [2], é possível controlar praticamente qualquer tipo de aparelho — TV, luzes, ar condicionado, *home theater* e outros — através de um dispositivo móvel, como celulares e *tablets*.

Há anos existem aparelhos controlados por controles remotos que emitem sinais de rádio frequência, como ventiladores de teto ou portões de garagens. Porém, esses aparelhos não são facilmente incluídos nos projetos de automação residencial, justamente por operarem via sinais de rádio frequência.

Este trabalho propõe a integração de um aplicativo móvel, um back-end em sistema embarcado e uma placa que transmite e sinais de rádio frequência. O MVP (Minimum Viable Product, ou, em português, Produto Minimamente Viável) desse produto foca no controle de ventiladores através de um aplicativo para dispositivos móveis. Em linhas gerais, ao pressionar um botão no aplicativo, um sinal é enviado para o sistema embarcado, que se comunica com a placa de rádio frequência que, por sua vez, envia o comando para o ventilador.

Nos Estados Unidos, público inicial do produto, cerca de 80 milhões de casas possuem pelo menos um ventilador de teto, e 25% de todas as famílias utilizam quatro ou mais ventiladores [3].

A empresa Altus-PCB possui *know-how* sobre tecnologias de transmissores e receptores de rádio frequência, mas precisava de uma empresa para terceirizar o desenvolvimento da aplicação móvel, back-end em sistema embarcado e API de comunicação entre os componentes. Assim, a proposta deste trabalho é o desenvolvimento dos itens necessários para controlar um ventilador de teto através de um aplicativo móvel, exceto pela tecnologia de rádio frequência, desenvolvida pela empresa contratante.

1.2 Objetivos

Este trabalho teve como objetivo o desenvolvimento de um aplicativo móvel, a criação de um sistema de back-end em sistema embarcado, a comunicação entre aplicativo e back-end e a comunicação entre o sistema embarcado e a placa de rádio frequência desenvolvida pela empresa Altus-PCB.

Todas essas etapas visam permitir que um usuário, através de um aplicativo em seu iPad, controle aparelhos comandados via sinais de rádio frequência. A escolha pelo MVP em versão para iPad foi definida pela empresa Altus PCB e uma adaptação para iPhone será desenvolvida em implementações futuras.

1.3 Estratégia de Solução

A ideia inicial da empresa Altus-PCB era o desenvolvimento de uma API de comunicação entre o aplicativo e a MCU (Microcontroller Unit, ou, em português, microcontrolador) da placa de rádio frequência, porém a mesma não obteve sucesso em contratar um profissional qualificado para lidar com tais tecnologias, principalmente pelas limitações de tempo para o lançamento do MVP.

A solução proposta pela Cheesecake Labs foi a utilização de um sistema embarcado intermediário, que pudesse rodar um sistema de back-end capaz de se comunicar com o aplicativo através da rede Wi-Fi, próximo de como normalmente os aplicativos se comunicam com sistemas web. Por fim, o sistema embarcado iria se comunicar com a placa de rádio frequência através de uma comunicação serial (ver Figura 1).

Dentre as tecnologias desenvolvidas, esse trabalho apenas não envolve a participação no desenvolvimento da placa de rádio frequência ou programação da MCU, destacados em amarelo. Por isso, não serão mencionados detalhes de implementações ou tecnologias utilizadas para esse componente do projeto.

Vale destacar a nomenclatura definida para o produto Bond, que inclui o sistema embarcado Raspberry Pi e a placa de rádio frequência mais MCU. Contudo, o aplicativo móvel também é chamado de Bond, ou aplicativo Bond.

Por fim, é importante esclarecer que em uma rede, ou no contexto do foco do produto, em uma casa, é possível ter múltiplos Bonds que atuam em múltiplos

dispositivos (como ventiladores) e que podem ser acessados por múltiplos aparelhos móveis, como tablets.

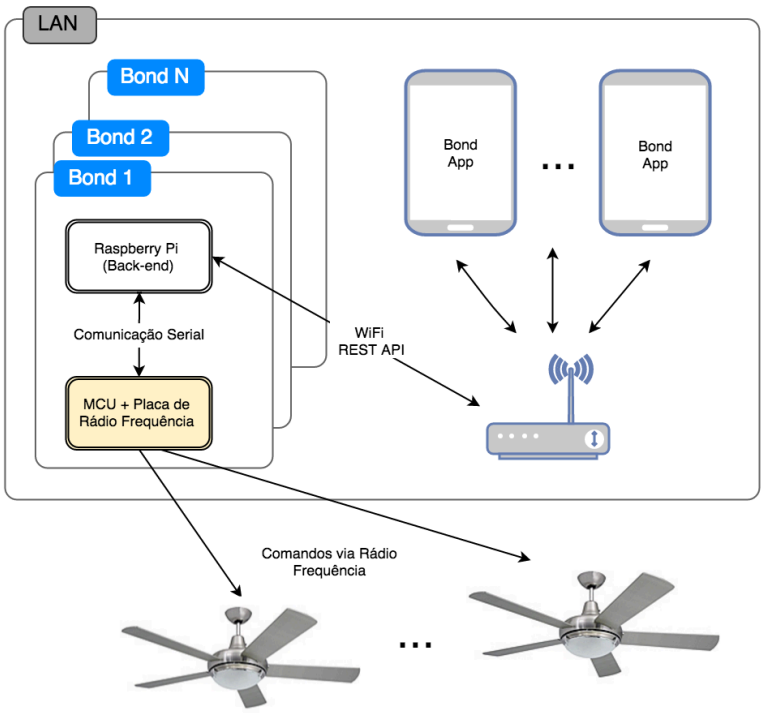

Figura 1 – Estratégia de Solução.

1.4 Estrutura do Documento

No capítulo 2 serão descritos, brevemente, as duas empresas envolvidas nesse projeto, a Cheesecake Labs e a Altus PCB, assim como a forma de colaboração entre ambas.

No capítulo 3 serão descritas as tecnologias utilizadas durante a realização desse projeto, incluindo o desenvolvimento de aplicativos para iOS, a interface de comunicação, o conceito de back-end e o mesmo em sistemas embarcados, linguagens de programação e, por fim, a descoberta automática de dispositivos.

No capítulo 4 será apresentado o produto na perspectiva do usuário, abordando detalhes pontuais de implementação quando necessários.

No capítulo 5 serão apresentadas as implementações, em um nível de abstração razoável para o entendimento dos passos percorridos para a concretização desse trabalho e assim como de seu nível técnico.

No capítulo **Error! Reference source not found.** é feita uma breve descrição dos resultados esperados e alcançados, apontando gargalos que poderão ser alvos de melhorias em esforços futuros.

Por fim, no capítulo 6 será apresentada a conclusão e perspectivas do produto desenvolvido.

2 A Empresa

2.1 Cheesecake Labs

Figura 2 – Logo Cheesecake Labs.

A Cheesecake Labs (Figura 2) foi fundada em dezembro de 2013 por quatro ex-alunos do curso de Engenharia de Controle e Automação – UFSC, que trabalhavam juntos desde 2008. Em 2012, após se mudarem para morar juntos e trabalhar em um aplicativo iOS chamado Dyfocus, Alexandre, Cássio, Marcelo e Victor conseguiram atrair aproximadamente dez mil usuários em menos de um ano. Com essas experiências juntos, os quatro decidiram empreender em uma área onde o modelo de negócio fosse mais conhecido.

Como Victor fez importantes contatos em experiências no Vale do Silício, Califórnia, incluindo estágio na empresa Uber, os quatro começaram a desenvolver produtos para empresas nos EUA. Após trabalhar com empresas dos EUA por um tempo, os ex-alunos e agora sócios, decidiram que desenvolver aplicações e tornar as ideias de terceiros uma realidade, não era tão animador quanto trabalhar em suas próprias ideias, por isso eles repensaram o modelo de colaboração remota.

Nesse momento, eles decidiram trabalhar apenas em projetos inovadores e que motivam os desenvolvedores a utilizarem tecnologias interessantes. Com isso, não apenas os *Cakers* passaram a trabalhar em projetos motivadores, como também começaram a fechar planos de ações com as empresas que prestavam serviços, além de disponibilizar planos de ações com os desenvolvedores.

Dessa forma, todos os envolvidos na concepção de um produto se sentem parte e donos do mesmo, como realmente são, dedicando-se ao máximo para o sucesso do projeto, desde o MVP, até uma aplicação robusta e escalável.

Com pouco mais de 2 anos, a Cheesecake Labs tem mostrado resultados expressivos [4], com sua equipe aumentando de quatro, para trinta membros.

2.2 Altus PCB

Figura 3 – Logo Altus PCB.

A empresa Altus PCB (Figura 3), localizada em Cresskill, Nova Jérsei, EUA, é especializa em projetos e entrega de PCBs (Printed Circuit Board, ou, em português, Placa de Circuito Impresso), suporte técnico e serviços para empresas de OEM (Original Equipment Manufacturers, ou, em português, Fabricante de Equipamento Original) e EMS (Electronic Manufacturing Services, ou, em português, Serviços de Manufatura Eletrônica) em todo o mundo. As soluções podem ser adaptadas para os clientes, com o objetivo de otimizar o preço, qualidade e entrega para cada placa.

Devido ao *know-how* de projetos eletrônicos, além de ser formado em Engenharia Elétrica, o fundador da empresa teve a ideia de desenvolver um módulo transmissor e receptor de sinais de rádio frequência, incluindo um sistema para armazenar e configurar novos sinais. Essa solução se encaixa em muitos projetos, entre eles no trabalho em questão.

Como o desenvolvimento de aplicativos não fazia parte das suas especialidades, a Altus PCB decidiu procurar por uma empresa parceira, para desenvolver o software da aplicação.

2.3 Colaboração Remota

Para a colaboração e desenvolvimento do projeto remotamente, foram estabelecidos alguns canais de comunicação, um sistema de versionamento de códigos e ferramentas de gerenciamento de atividades:

- Comunicação: canal contínuo via Slack [5] e reuniões semanais via Hangouts [6];
- Versionamento: repositórios privados na conta da empresa Altus PCB na plataforma GitHub [7];
- Gerenciamento de Atividades: acompanhamento via software Pivotal Tracker [8].

3 Tecnologias Utilizadas

Este capítulo possui como objetivo apresentar algumas das tecnologias e conceitos utilizados no decorrer deste trabalho, procurando facilitar o entendimento do que será exposto nos capítulos posteriores. Uma análise das dificuldades encontras na utilização dessas tecnologias é feita ao final do capítulo.

3.1 Desenvolvimento de Aplicativos iOS

O sistema operacional iOS (antes chamado de iPhone OS) é um sistema desenvolvido pela Apple Inc. originalmente criado para o dispositivo portátil iPhone e atualmente também utilizado no iPod Touch e no iPad. O mesmo sistema ainda deu origem a outros sistemas operacionais, como o watchOS destinado ao Apple Watch, e ao tvOS, destinado à Apple TV.

O iOS gerencia o hardware do dispositivo e fornece as tecnologias necessárias para programar aplicativos nativos. A arquitetura do iOS consiste em quatro camadas de abstração [9]: a camada Core OS, a Core Services, a Media e a camada Cocoa Touch, como ilustrado na Figura 4.

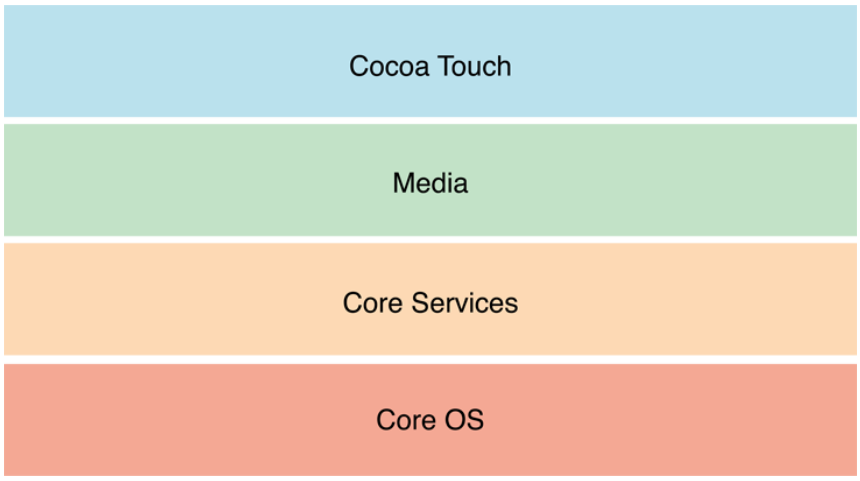

Figura 4 – Camadas de abstração do sistema iOS.

As camadas inferiores contêm serviços e tecnologias fundamentais. As camadas superiores, criadas sobre as camadas mais baixas constituem-se de serviços e tecnologias mais sofisticadas. Cada uma delas oferece um conjunto de frameworks que podem ser utilizados durante o desenvolvimento.

3.1.1 Interface de Desenvolvimento Xcode

O Xcode é o ambiente de desenvolvimento integrado (IDE) da Apple, constituído de um editor de código, interface gráfica do usuário, e outros recursos que auxiliam o desenvolvimento de aplicativos iOS, macOS, watchOS e tvOS. O iOS SDK, presente no Xcode, inclui as ferramentas, compiladores e estruturas necessárias para o desenvolvimento de aplicações iOS.

A IDE é ilustrada na Figura 5 onde é possível visualizar a área de navegação na esquerda, a seção de debug no centro inferior, a área de utilitários na direita e por fim, a seção de desenvolvimento no centro.

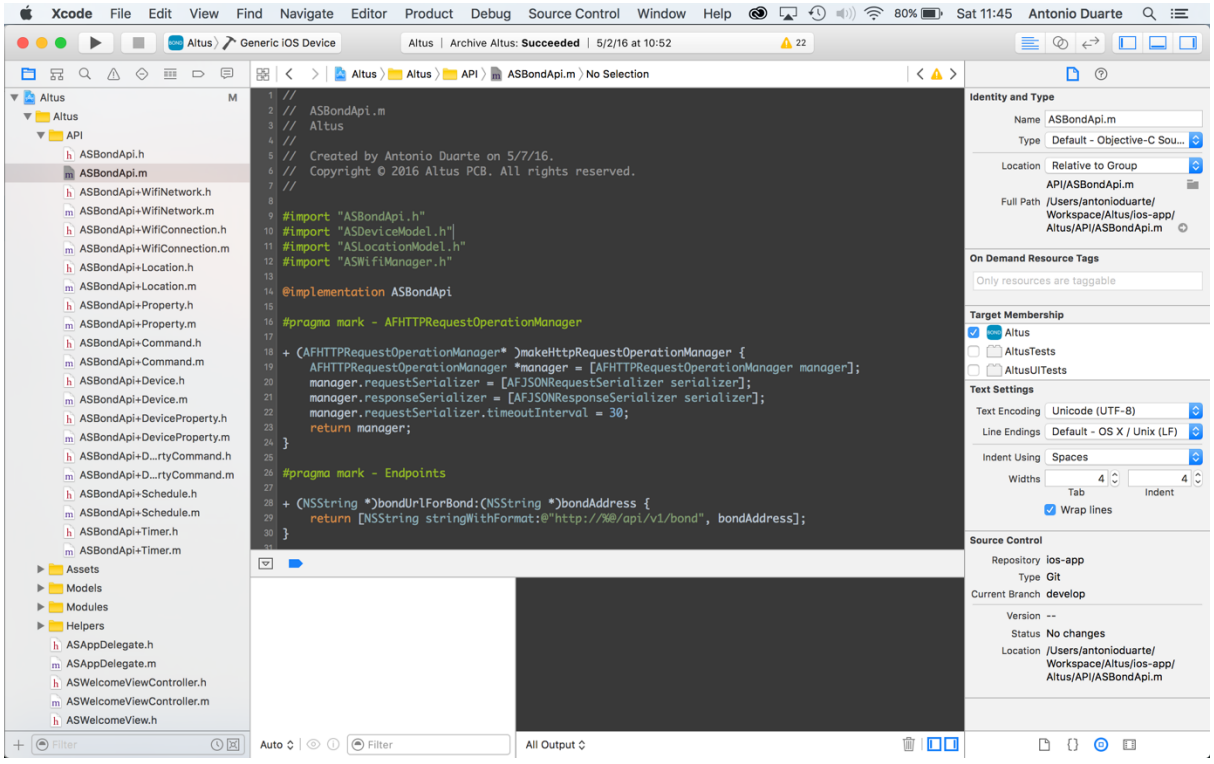

Figura 5 – Interface de Desenvolvimento Xcode.

3.1.2 Linguagens de Programação

A linguagem utilizada oficialmente até o iOS 7 era o Objective-C [10], que sempre foi um degrau mais alto na escalada pela proficiência, tanto para o desenvolvedor que vinha de outras linguagens, quanto para os iniciantes no mundo da programação.

Em 2014, a Apple Inc. lançou uma nova linguagem de programação, chamada de Swift [11] e desde o iOS 8 as aplicações podem ser desenvolvidas com

Objective-C ou Swift, inclusive combinando as duas no mesmo projeto. Apesar disso, as empresas de desenvolvimento de software possuem uma postura mais conservadora em relação a adoção de novas linguagens, principalmente devido ao suporte e a complexidade das aplicações — que utilizam diversas bibliotecas escritas ao longo dos anos, não compatíveis com a nova linguagem. O fato da linguagem ainda não ser consolidada, também acarreta em atualizações da linguagem com muitas alterações, o que exige muitos esforços para manter a compatibilidade do projeto.

Por esses motivos, a linguagem de programação Objective-C foi adotada para o presente projeto.

3.1.3 Gerenciador de Dependências

Figura 6 – Logo CocoaPods.

Devido a dificuldade de importar bibliotecas e disponibilizá-las no Xcode, foi utilizado um gerenciador de dependências nomeado CocoaPods, uma ferramenta que vem se tornando padrão nos projetos de aplicativos iOS, com mais de dezoito mil bibliotecas disponíveis [12]. As principais bibliotecas utilizadas para este trabalho são listadas a seguir:

- AFNetworking: destaque na comunidade de desenvolvedores devido à sua API moderna e ativa comunidade, é feita sob a fundação de networking do próprio sistema operacional, tirando proveito do que há de melhor para trabalhar com comunicação de rede, tendo por padrão inclusive classes para lidar com conteúdo XML e JSON.

- Crashlytics: criado para diminuir o tempo de detecção, acesso e correção de bugs, essa biblioteca além de identificar falhas no aplicativo, é capaz de isolar a causa raiz até a linha exata de código, além de informar as falhas em tempo real.

- MagicalRecord: Core Data é a solução built-in da Apple para a persistência e consulta de dados no iOS. Embora a Apple constantemente tente torna-lo mais fácil de utilizar, o Core Data continua sendo uma tecnologia difícil de dominar, inclusive pela quantidade de código necessária para executar operações simples. A biblioteca MagicalRecord fornece métodos convenientes que envolvem códigos clichês comuns a todos os métodos do Core Data, criando uma camada de abstração acima e facilitando o manuseio do Core Data.

- Google Analytics: Essa biblioteca fornece ferramentas de medição, permitindo identificar quais partes do aplicativo são utilizadas com mais ou menos frequência, tempo de duração, número de usuários ou sessões entre outras métricas. O principal objetivo é ter uma ideia da experiência do usuário ao utilizar o aplicativo.

3.2 REST API

Uma REST API é uma interface de programação de aplicativos que segue a arquitetura REST, uma abstração da arquitetura World Wide Web, mas precisamente, é um estilo arquitetural que consiste de um conjunto coordenado de restrições arquiteturais aplicadas a componentes, conectores e elementos de dados dentro de um sistema de hipermídia distribuído [13]. O REST usa integralmente as mensagens HTTP para se comunicar através do que é definido no protocolo sem precisar recriar protocolos específicos para cada aplicação. As características principais são:

- Utiliza o protocolo HTTP (verbos, *accept headers*, códigos de estado HTTP, *Content-Type*) de forma explícita e representativa para se comunicar. URIs são usados para expor a estrutura do serviço;
- Não possui estado entre essas comunicações, ou seja, cada comunicação é independente e uniforme (padronizada) precisando passar toda informação necessária;
- Facilita o cache de conteúdo no cliente;
- Possui clara definição do que faz parte do cliente e do servidor. O cliente não precisa saber como o servidor armazena dados, por exemplo. Assim cada implementação não depende da outra e se torna mais escalável;

- Permite o uso em camadas também facilitando a escalabilidade, confiabilidade e segurança;
- Frequentemente é criado com alguma forma de extensibilidade.

Toda API possui uma notação comum para transferência de dados, onde as mais comuns são XML (Extensible Markup Language) e JSON (JavaScript Object Notation). Dessa forma, basta cada linguagem fornecer uma maneira de transformar dados no formato XML, ou JSON, em objetos nativos na linguagem e vice-versa. Isso permite que duas aplicações de linguagens de programação e plataformas diferentes possam se comunicar através de um formato definido. No presente trabalho o formato JSON foi escolhido por ser a opção mais comum em aplicações web e móveis, principalmente pela sua simplicidade.

A seguir será apresentada a forma de documentação da API que possibilita que terceiros possam consultar a interface de comunicação e analisar os esforços necessários para integração.

3.2.1 Apiary

Figura 7 – Logo Apiary.

Uma das grandes dificuldades na documentação da API é a ilustração de sua utilização para o desenvolvedor, principalmente aqueles que estão tendo o primeiro contato com a mesma. Ao iniciar a integração, é natural que os desenvolvedores queiram saber como suas requisições HTTP estão sendo recebidas pela API, assim como as suas respostas.

A plataforma Apiary utiliza a linguagem de documentação nomeada API Blueprint [14], baseada em *Markdown*, o que normalmente é vista como uma vantagem para diminuir a curva de aprendizado. Além do design da documentação, a Apiary se propõe em cuidar de aspectos de relacionamentos de ciclo de vida da API, através de ferramentas que possibilitam fomentar e gerar uma suíte de testes para a mesma.

Por fim, a Figura 8 ilustra um dos métodos do presente trabalho descrito com a linguagem API Blueprint e na Figura 9 é possível visualizar o resultado da documentação gerada.

```
# Group Bond

## Bond Settings [/api/v1/bond/]

List the available settings for the Bond (this only works locally,
since it is used for finding if a given ip belongs to a Bond server).
The remote equivalent would be /api/v1/bonds/{bond_id}/.

### Getting information from Bond [GET]

+ Response 200 (application/json)

        {
            "id": "SUD8U91",
            "temperature": 32.5,
            "illumination": 0.8,
            "humidity": 0.55,
            "location": {
                "id": 12938,
                "name": "Kitchen"
            }
        }
```

Figura 8 – Exemplo de documentação utilizando API Blueprint.

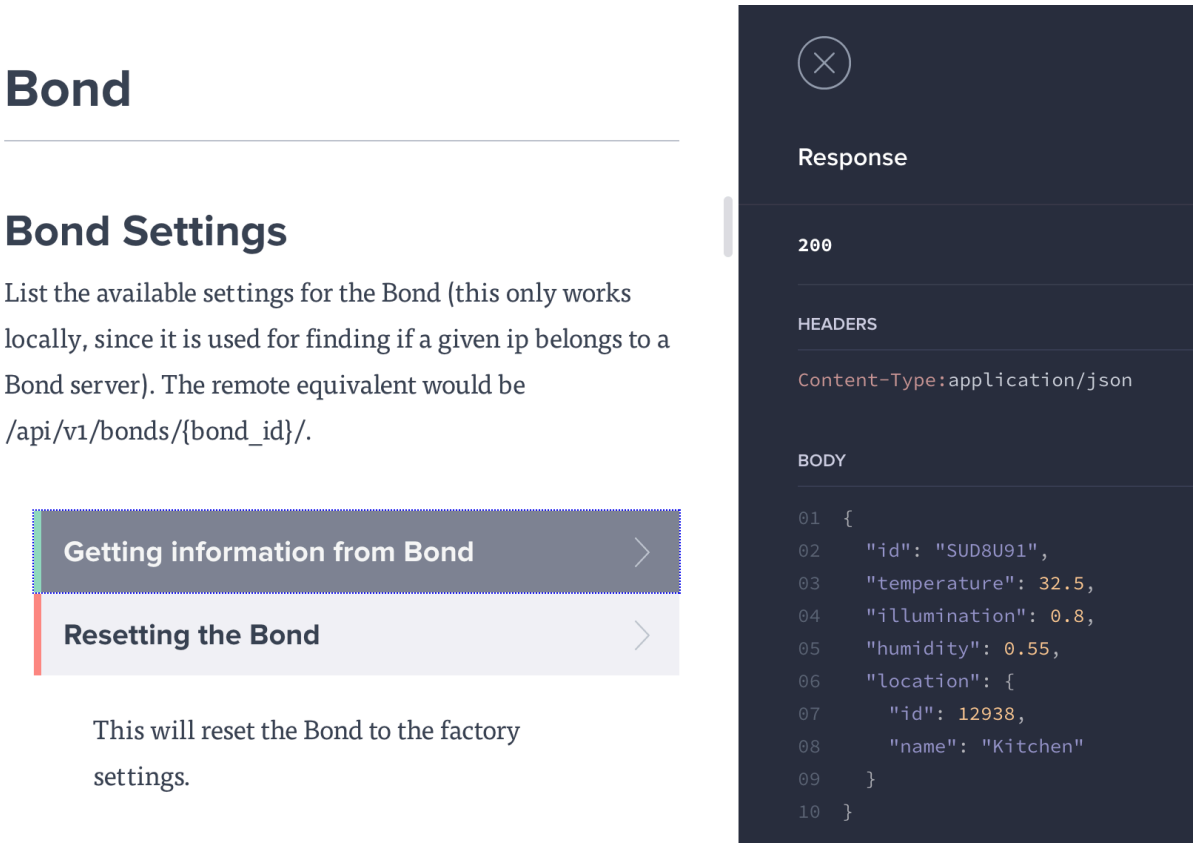

Figura 9 – Exemplo de documentação gerada pela Apiary.

3.3 Back-end

Ao navegar por um website, ou por um aplicativo móvel, não é evidente que as aplicações são desenvolvidas e atuam por duas etapas distintas: o front-end e o back-end.

Em linhas gerais o front-end é responsável pela apresentação de conteúdos, formulário de entradas e tratamento das ações dos usuários, sendo uma interface entre usuário e aplicação no qual se encontram botões, imagens, textos, logos, animações e etc. O back-end é responsável pelo armazenamento das informações em banco de dados, da maioria das lógicas da aplicação e da maioria da infraestrutura. É comum em aplicativos móveis que o back-end esteja na web, disponível para múltiplos clientes, permitindo que uma pessoa acesse as mesmas informações de diferentes dispositivos e possibilitando interações entre usuários ou aplicações.

Existe uma classificação mais detalhista, onde um aplicativo é dividido em front-end, focado na codificação do design da aplicação, um back-end do front-end,

responsável pelas lógicas de formulários, transições e tratamento dos elementos de interface em geral, e o back-end, mencionado anteriormente, responsável pelo banco de dados, integrações com serviços e lógicas gerais. Em uma aplicação web essa divisão é facilmente notada pelas diferentes tecnologias utilizadas, como JavaScript, CSS e HTML para o front-end, frameworks como AngularJS e ReactJS para o back-end do front-end e por fim, tecnologias como Django, Ruby on Rails ou NodeJS, para o back-end propriamente dito. Já em aplicativos iOS essa divisão pode ser feita pela utilização do *Interface Builder* ou *Storyboards* para o front-end, Objective-C ou Swift para o back-end do front-end e os frameworks como Django, Ruby on Rails ou NodeJS para o back-end web.

Em ambos os casos, a comunicação entre o back-end, responsável pelo armazenamento, servidores e lógicas gerais, é feita através de uma integração via API de comunicação (Seção 3.2), onde o back-end permanece completamente desacoplado do front-end — ou do back-end do front-end.

Neste projeto, como o back-end é hospedado em um Raspberry Pi Zero, com limitações de sistema e hardware, foi definido que o framework Node.js era apropriado como tecnologia de back-end, sendo mais leve e ao mesmo tempo com as mesmas capacidades de seus competidores [15] [16], como Django ou Ruby on Rails.

Nas seções a seguir são apresentados o sistema embarcado Raspberry Pi Zero, o framework NodeJS e sua linguagem JavaScript, para melhor compreensão das tecnologias utilizadas para o back-end deste trabalho.

3.3.1 Raspberry Pi Zero

A Pi Zero foi lançada pela Fundação Raspberry Pi em novembro de 2015, chamando a atenção pelo preço de apenas cinco dólares e dimensões muito reduzidas (Figura 10). Apesar disso, possui capacidade comparável com as outras placas de tamanho e custos muito superiores.

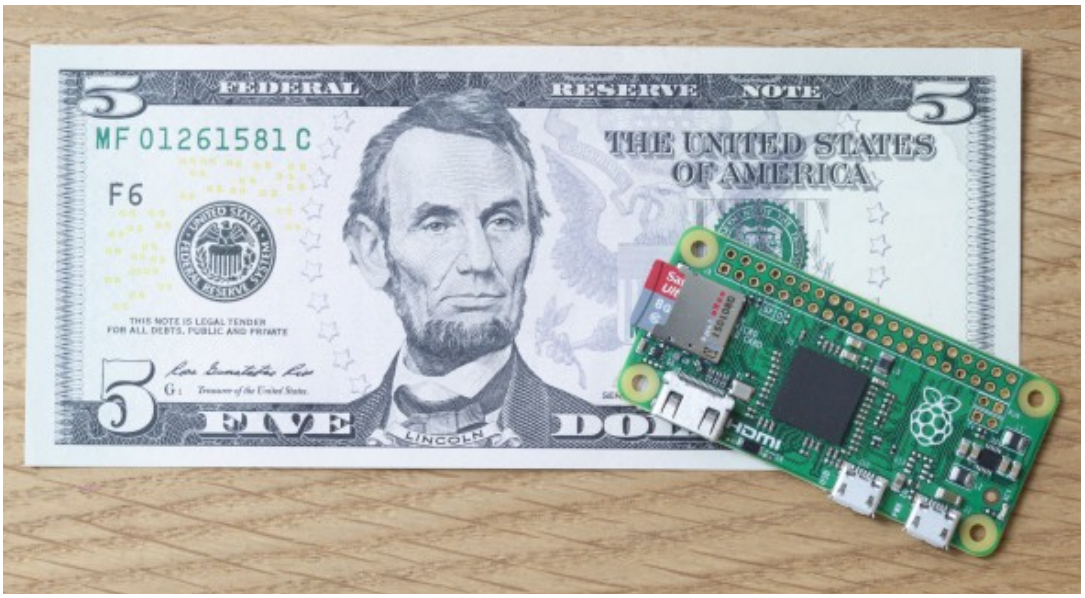

Figura 10 – Comparação entre Raspberry Pi Zero e uma cédula de cinco dólares. A cédula também representa o custo da placa eletrônica.

A seguir são apresentadas as características da Raspberry Pi Zero:

- Processador Broadcom BCM2835, 1 GHz, ARM11;
- 512MB de memória LPDDR2 SDRAM;
- Slot para memória Micro SD;
- Conector mini-HDMI com saída de vídeo de 1080p60;
- Dois conectores micro-USB, sendo um para alimentação e outro para dados;
- 40 pinos de entrada e saída;
- Dimensões reduzidas: 65 mm x 30 mm x 5 mm.

Como a placa não conta com um módulo WI-FI, necessária para a comunicação na rede, foi utilizado um Dongle Wi-Fi USB. Apesar disso, e dessa tecnologia não ser uma proposta comparável com a ideia inicial de utilizar apenas uma MCU, principalmente em termos de custos de produção em escala, a Pi Zero foi escolhida para servir de ponte entre o aplicativo iOS e a placa de rádio frequência, como proposta para o MVP e também devido o custo e tamanho comparados com outras placas com as mesmas capacidades.

Como solução para o MVP, essa proposta atendeu as exigências de prazo de desenvolvimento, visto que o tempo de desenvolvimento de uma API e back-end para um sistema embarcado que roda sistemas de mais alto nível, como Node.js, é significante menor do que o desenvolvimento em linguagens de mais baixo nível e

um ambiente próximo do metal. Contudo, essas tecnologias não serão utilizadas em produção, devido ao custo elevado em comparação com sistemas mais simples, a exemplo de uma MCU.

3.3.2 NodeJS

Figura 11 – Logo NodeJS.

O framework Node.js possui uma curva de aprendizagem relativamente pequena por utilizar JavaScript como linguagem de programação, porém sua característica mais importante é utilizar um modelo de entrada e saída de dados direcionados a eventos não bloqueantes (do Inglês, *non blocking I/O*). Por isso, diferente do comportamento tradicional das tecnologias de programação, as requisições feitas ao Node.js não permanecem presas ao processo até sua conclusão.

Essas características tornam essa tecnologia leve e eficiente, ideal para aplicações com troca intensa de dados [17], incluindo tecnologias com recursos limitados, como pouca memória e capacidade de CPU [18].

Com a criação de rotas no framework Node.js, é possível criar uma REST API (Capítulo 3.2) para receber os chamados de uma aplicação iOS (Seção 3.1) e interagir com o banco de dados hospedado no sistema embarcado.

3.3.3 Javascript

Figura 12 – Logo JavaScript.

Javascript é uma linguagem de programação criada em 1995 por Brendan Eich enquanto trabalhava na Netscape Communications Corporation [19]. É uma linguagem de alto nível, dinâmica, não-tipada e interpretada, sendo uma das três tecnologias mais utilizadas para a produção de conteúdos para web – as outras duas são HTML e CSS.

Embora originalmente projetada para rodar em navegadores, no lado do cliente, atualmente Javascript também participa na execução de aplicações no lado do servidor, sendo a linguagem de frameworks de back-end, como Node.js.

3.4 Bonjour

Figura 13 – Logo Bonjour.

Conhecido inicialmente como Rendezvous, o Bonjour é um serviço de autoconfiguração implementado pela Apple Inc. que usa conceitos de Zeroconf (do Inglês, Zero Configuration Networking), que é um conjunto de técnicas que permitem criar uma rede IP sem necessitar de configuração ou servidores, tudo automaticamente [20]. Entre os componentes que o serviço permite acessar estão

computadores, impressoras, dispositivos e serviços em nuvem. Essa tecnologia está presente no Windows e MacOS através do Bonjour e no Linux, via Avahi [21].

Na prática, o Bonjour detecta tudo o que for necessário para configurar uma rede, pois ele permite a descoberta automática de dispositivos e serviços em uma rede local utilizando apenas o protocolo IP padrão.

No contexto deste trabalho, o Bonjour foi utilizado tanto na aplicação iOS, como no back-end presente em sistema embarcado, via Avahi, publicando serviços a serem encontrados. Dessa forma, através do aplicativo iOS é possível se conectar automaticamente aos sistemas embarcados presentes na mesma rede.

4 Bond

Este capítulo tem como objetivo apresentar o projeto Bond, passando uma ideia geral sobre o seu funcionamento do ponto de vista do usuário do aplicativo iOS, mencionando aspectos da implementação quando necessário.

4.1 Primeiro Uso

Ao entrar no aplicativo pela primeira vez em um dispositivo iOS (ver Figura 14), é possível tomar duas ações: conectar e configurar o seu primeiro Bond, ou encontrar um Bond previamente configurado.

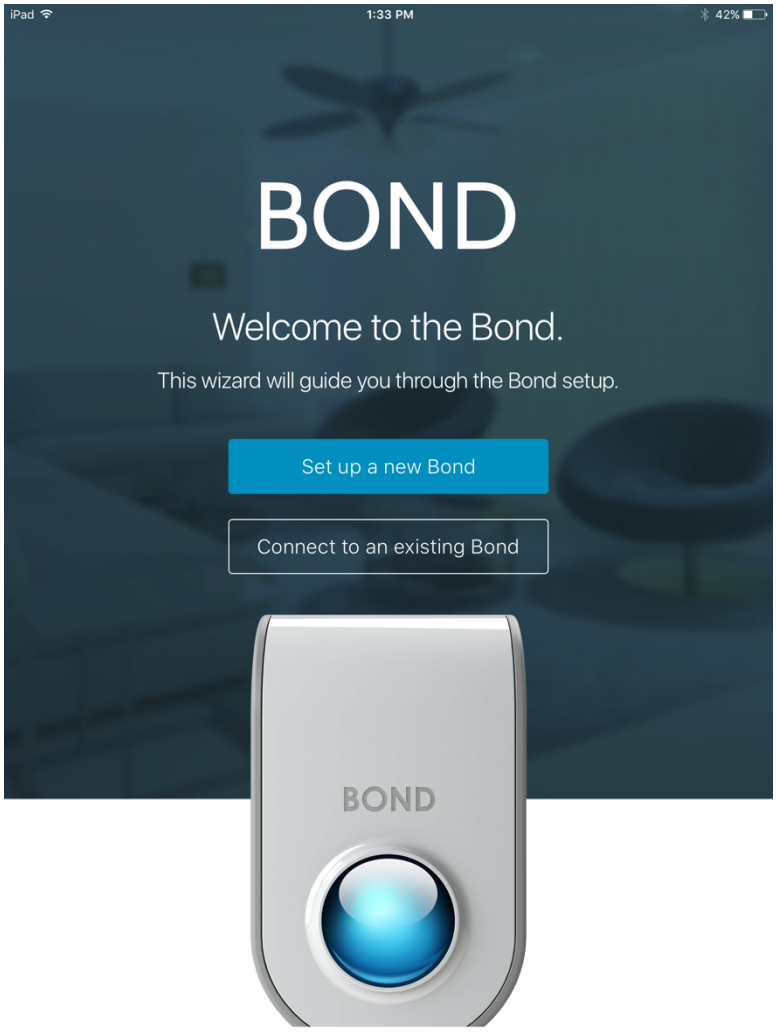

Figura 14 – Tela inicial do aplicativo Bond.

Para conectar e configurar um Bond é preciso se conectar ao mesmo e passar instruções para que ele se conecte em uma determinada rede. Depois disso, o dispositivo iOS e Bond passam a se comunicar através dessa rede, permitindo que o Bond esteja acessível via internet ou rede local.

A segunda opção permite encontrar um Bond que esteja na internet ou rede local utilizada pelo dispositivo iOS. Isso é possível pois o Bond é alcançável através da descoberta automática via Bonjour (Seção 3.4).

A seguir serão descritos os passos que o usuário deve percorrer para configurar um novo Bond, ainda em modo hotspot ou *access point* (do Inglês, ponto de acesso) – não conectado na rede local ou internet.

4.2 Configuração de Bonds

Primeiramente o usuário deve optar pela opção “Configurar um novo Bond” na tela inicial, ilustrada na Figura 14.

Em seguida, uma tela com informações básicas (ver Figura 15) instrui o usuário a certificar-se que o Bond está preparado para ser conectado. O mesmo deve estar ligado em uma fonte de energia e a luz azul deve indicar o modo hotspot, não conectado a nenhuma rede.

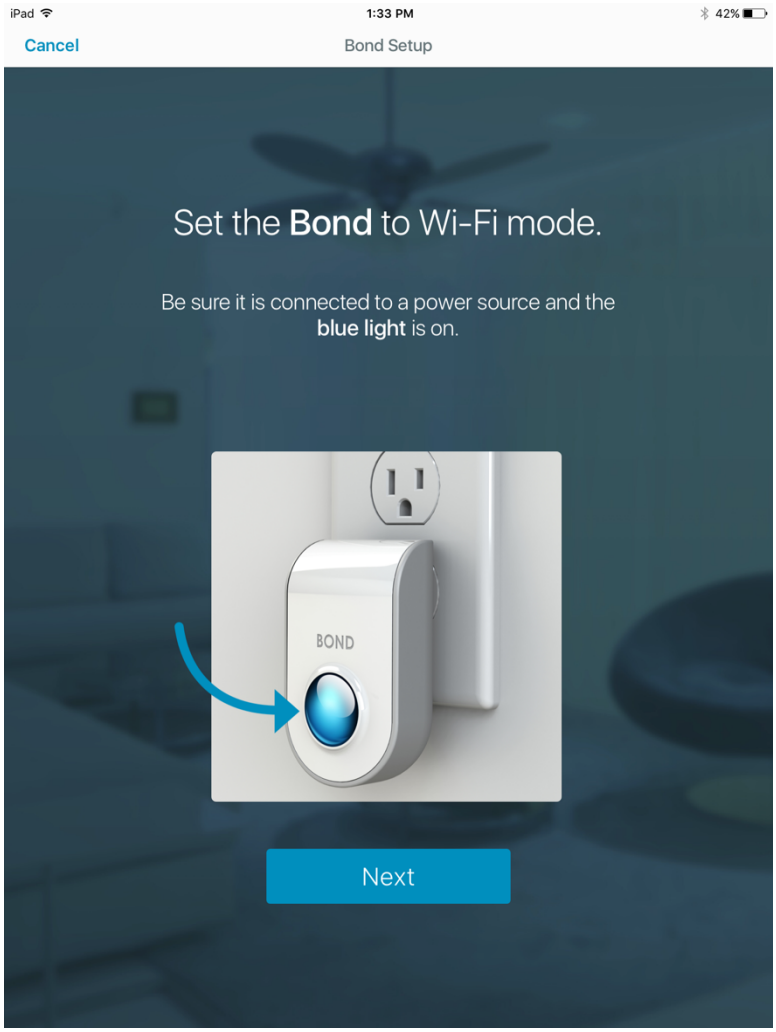

Figura 15 – Tela de instruções para configuração de um novo Bond.

Após certificar-se que o mesmo está em modo hotspot, e pressionar o botão de *Next*, a próxima tela instrui o usuário a ir nas configurações do dispositivo iOS e selecionar o ponto de acesso Wi-Fi do Bond como a opção de wireless (Figura 16). Entre as opções de wireless disponíveis, a opção do Bond é sempre descrita por "Bond ID", onde o ID é um número identificador único para cada Bond, a exemplo do Bond 9CMVNK3 ilustrado na Figura 17.

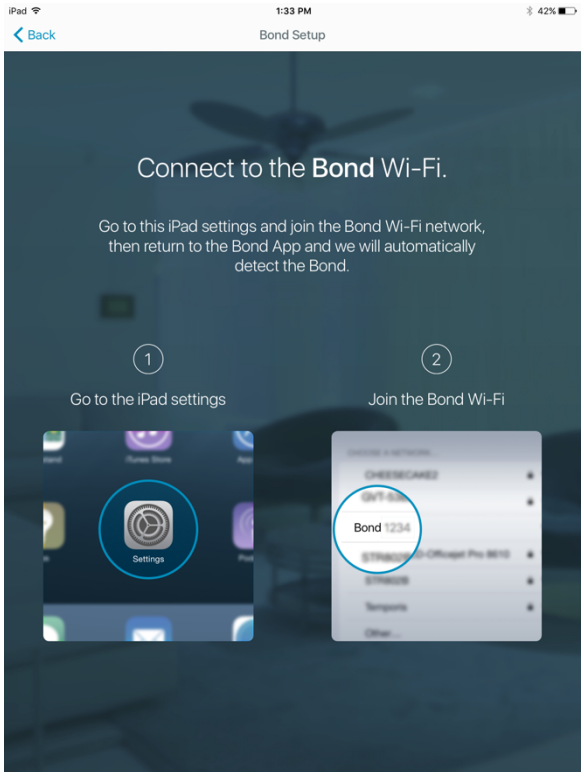

Figura 16 – Tela de instruções para conexão no ponto de acesso do Bond.

Figura 17 – Tela de configurações do iPad com Wi-Fi selecionada.

Ao retornar para o aplicativo, agora conectado ao Bond, o aplicativo irá automaticamente solicitar e listar as redes que estão ao alcance do Bond, como ilustrado na Figura 18. Selecionando a Wi-Fi desejada, é apresentado um campo para inserir a senha da mesma, para que o Bond possa se conectar (Figura 19).

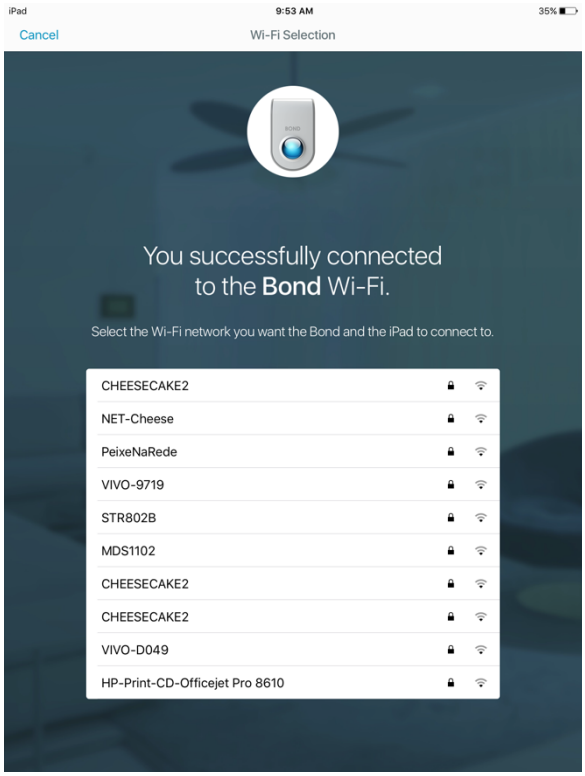

Figura 18 – Tela com a lista das redes ao alcance do Bond.

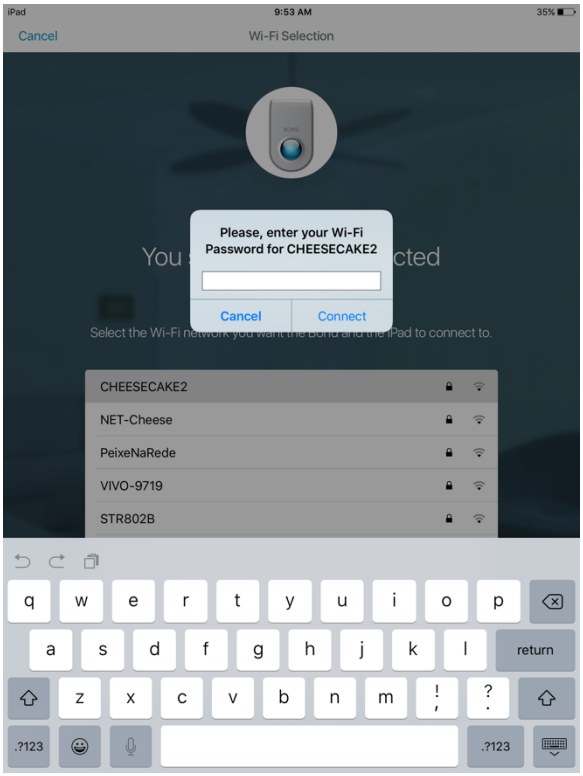

Figura 19 – Tela com campo para inserir a senha da Wi-Fi selecionada.

Nesse momento é preciso fazer alguns comentários sobre as limitações da atual implementação. Após selecionar uma Wi-Fi, inserir uma senha e pressionar o botão *Connect,* o Bond iniciará os passos necessários para sair do modo hotspot (ver Figura 20), perdendo a conexão com o dispositivo iOS e tentando se conectar na rede previamente selecionada.

Atualmente o Bond não retorna ao modo hotspot ao falhar na conexão da rede selecionada, por isso, se o usuário selecionar a opção errada ou inserir a senha da Wi-Fi erroneamente, o Bond deve ser reiniciado para o modo hotspot e os passos descritos anteriormente devem ser realizados novamente. Esse processo pode levar alguns minutos, visto que o Bond precisa reiniciar, habilitar o seu ponto de acesso e posteriormente ser alcançável pelo iPad. Por isso, para evitar a inserção da senha erroneamente e o retrabalho de percorrer os passos anteriores, principalmente durante o desenvolvimento — onde frequentemente um novo Bond foi configurado para validação — o campo de senha permite a visualização da senha como texto pleno.

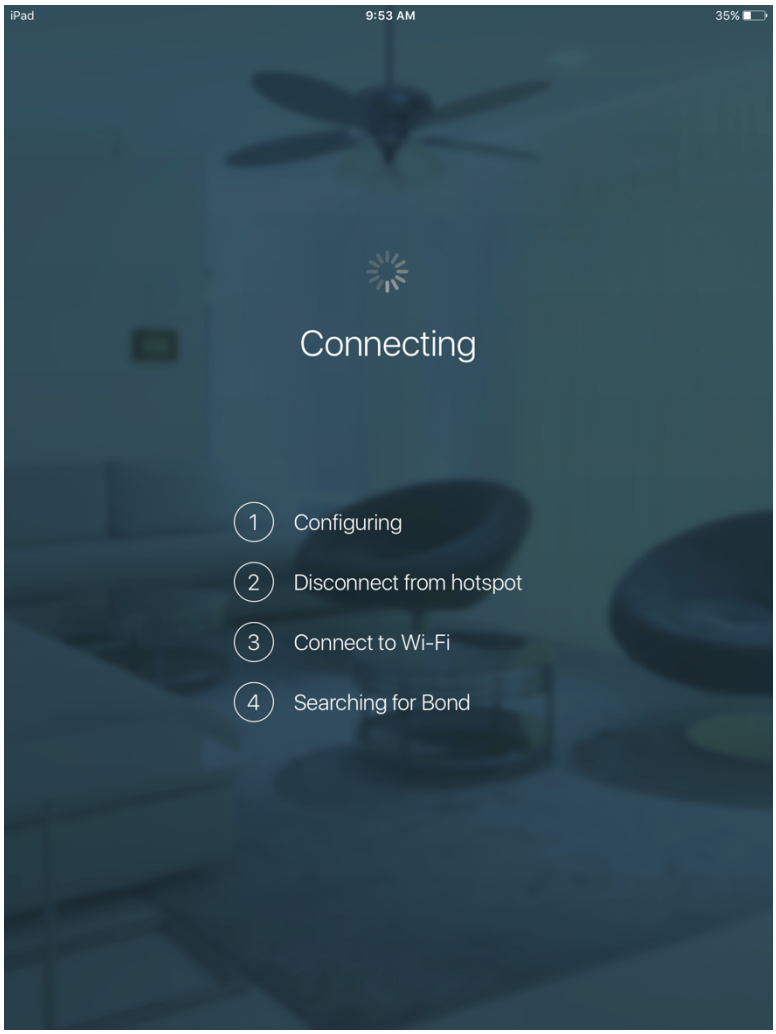

Figura 20 – Tela com os passos para a conexão do Bond na Wi-Fi selecionada.

Na Figura 20 é possível visualizar os quatro passos que o aplicativo percorre para começar a se comunicar através da rede Wi-Fi selecionada previamente. No primeiro são enviadas as informações da Wi-Fi e senha para o Bond. O segundo indica que o Bond se desconectou do iOS, pois está mudando do modo hotspot para se conectar da Wi-Fi selecionada. No terceiro passo, o aplicativo, não mais conectado ao hotspot do Bond, está se conectando a rede Wi-Fi selecionada previamente. No último, ambos os dispositivos iOS e Bond estão conectados na Wi-Fi selecionada e estão se encontrando na mesma, através da descoberta automática via Bonjour. Após os quatro passos, o aplicativo iOS e Bond passam a se comunicar através da rede Wi-Fi.

A configuração inicial de um novo Bond é a sua localização na residência, por isso, logo após conectar o Bond na rede, é preciso selecionar a localização do mesmo (ver Figura 21).

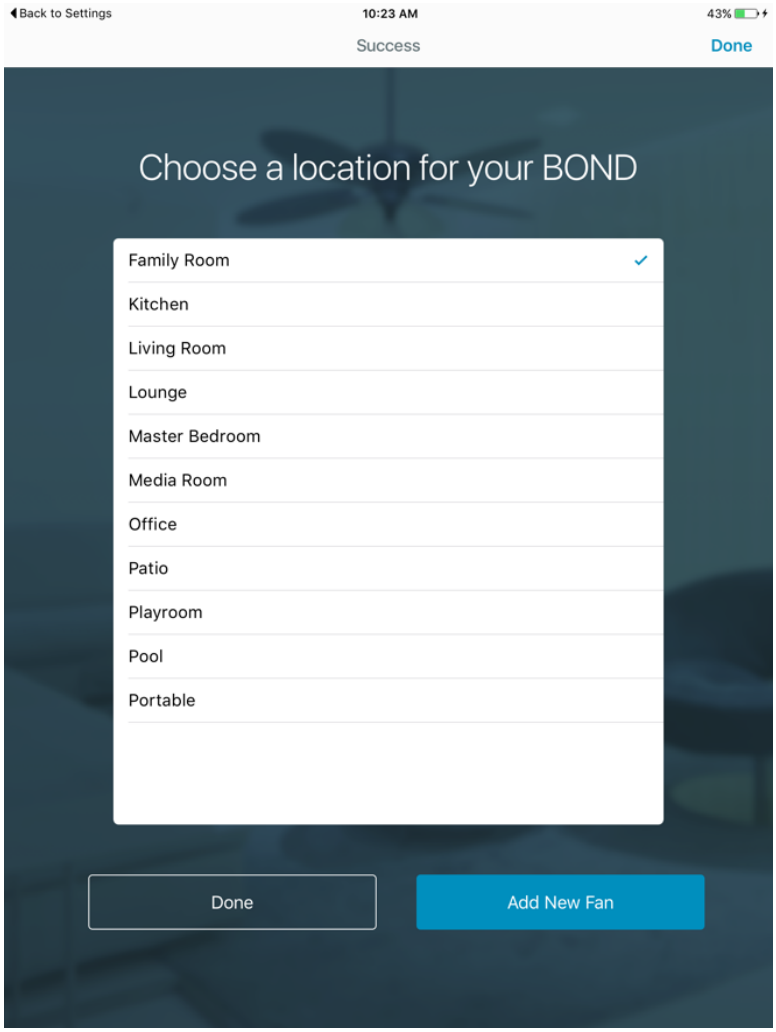

Figura 21 – Tela para selecionar a localização do Bond.

Em seguida, é dada a opção de configuração do primeiro dispositivo a ser controlado pelo Bond. Essa opção será descrita com detalhes na seção 4.4. Uma alternativa é configurar os dispositivos — na versão atual apenas ventiladores de teto — posteriormente, através do botão *Done* (do Inglês, pronto). Ao pressionar esse botão o usuário é direcionado para a tela que lista os Bonds configurados.

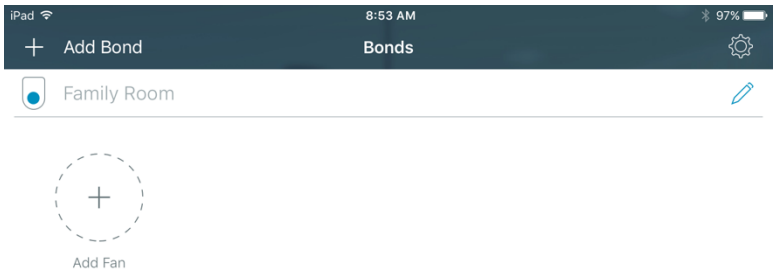

Figura 22 – Tela com a lista de Bonds contendo o primeiro Bond configurado.

Encerram-se assim os passos para a configuração do primeiro Bond, desde a sua conexão na rede local ou internet, até sua configuração inicial de localização e apresentação na lista de Bonds.

A seguir serão descritos os elementos e funcionalidades presentes na tela para listagem dos Bonds cadastrados.

4.3 Lista de Bonds

Após a configuração do primeiro Bond, a tela ilustrada na Figura 23, passa a ser a tela inicial do aplicativo. Para configurar um novo Bond, o usuário deve pressionar o botão *Add Bond* (do Inglês, adicionar Bond) no canto superior esquerdo da mesma. Ao fim dos passos descritos na seção 4.2, o mesmo será apresentado junto com os demais Bonds na mesma tela. A Figura 23 ilustra um exemplo da lista de Bonds contendo três Bonds configurados. Dois desses Bonds estão com uma maior opacidade, e uma mensagem sobre a conectividade *BOND NOT FOUDN ON THE NETWORK* (em Inglês, Bond não encontrado na rede). Essa mensagem avisa o usuário que determinados Bonds estão desconectados da rede onde o aplicativo está conectado e por isso, não são controláveis.

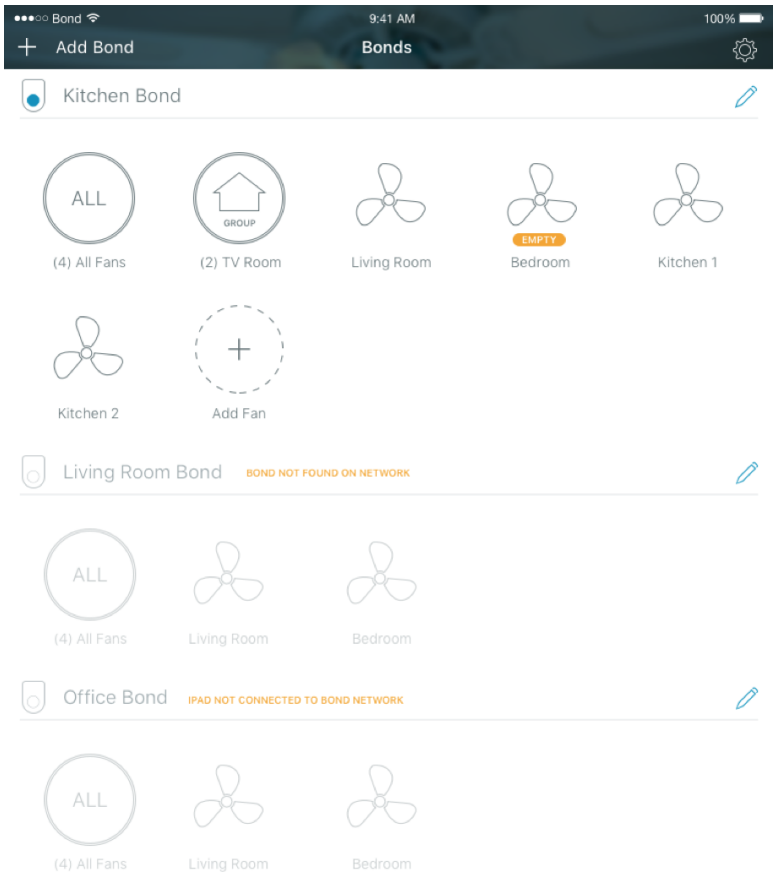

Figura 23 – Tela com lista de Bonds, contendo três Bonds e seus dispositivos.

No cabeçalho superior da Figura 23, no canto direito, é possível visualizar um ícone em formato de engrenagem que dá acesso às configurações gerais do aplicativo, ilustradas na Figura 24. A maioria dos elementos foram adicionados visando a versão final do aplicativo, como o sistema de cadastro de usuários, a compra de um novo Bond (*Buy New Bonds*), ou a recomendação do produto para amigos (*Tell a Friend*). Atualmente, essas funcionalidades não estão implementadas, porém a interface foi desenvolvida visando demonstrar o potencial do produto.

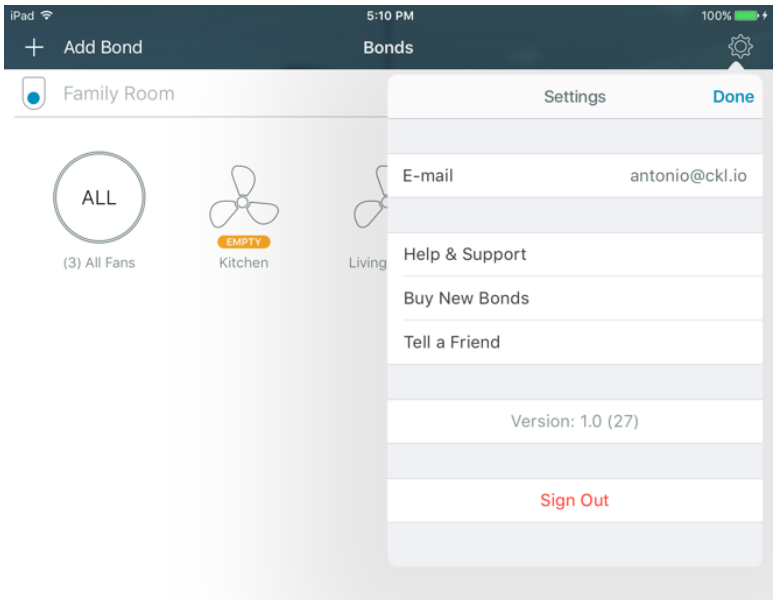

Figura 24 – Tela contento pop-up de configurações gerais.

Abaixo do cabeçalho é apresentada a lista de Bonds, onde para cada Bond é possível visualizar a sua localização no canto superior esquerdo e clicar no botão de configuração do Bond no canto superior direito, ambos indicados na Figura 25.

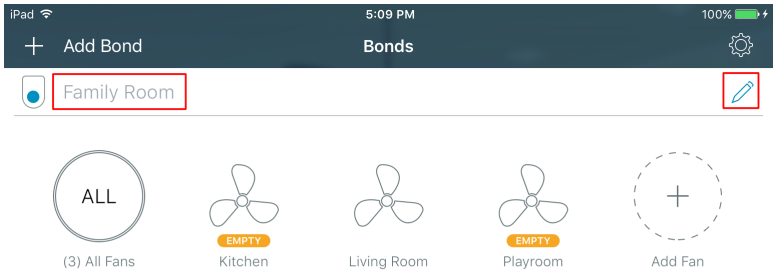

Figura 25 – Bond listado e sua localização e ícone para configurações.

Ao pressionar o botão de configuração, ícone com formato de lápis, é possível alterar a localização, manusear os ventiladores cadastrados e visualizar as informações do Bond (Figura 26). Clicando em Location (do Inglês, localização), uma lista com opções de localizações abre para seleção pelo usuário (Figura 27). Clicando em Fan (do Inglês, ventilador), uma lista com os ventiladores cadastrados no Bond é apresentada (Figura 28), sendo possível remover um ou mais ventiladores configurados através do ícone em vermelho, do lado esquerdo de cada dispositivo.

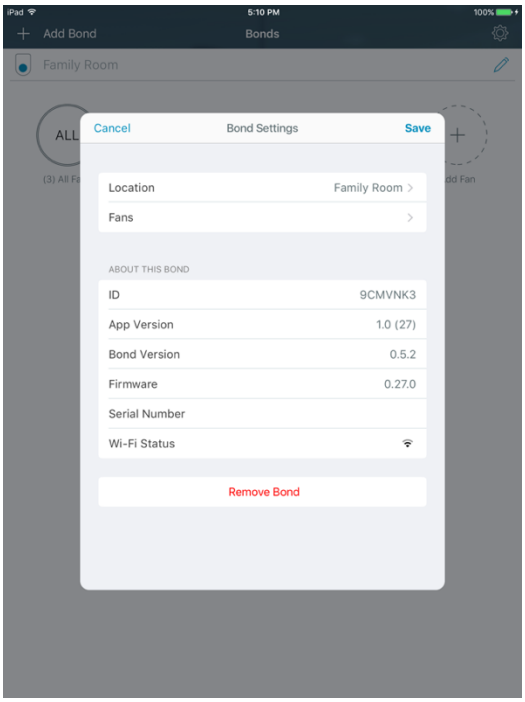

Figura 26 – Tela contento pop-up de configurações para cada Bond.

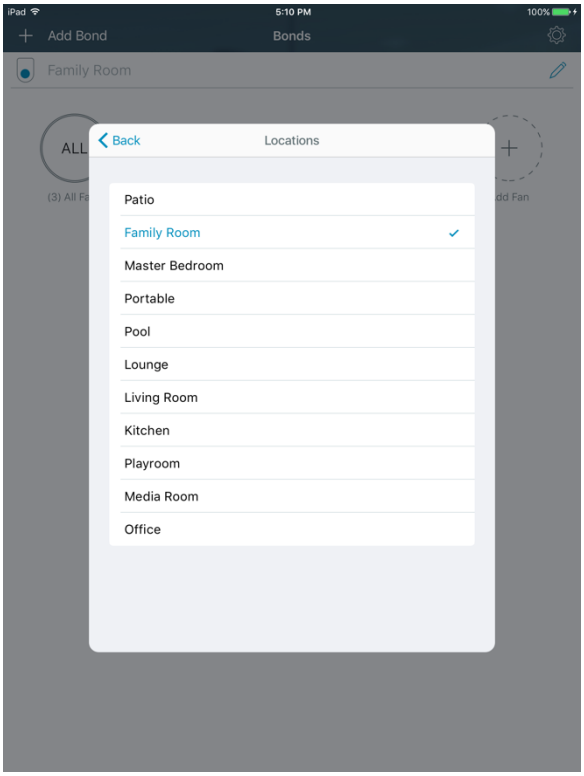

Figura 27 – Lista de localizações nas configurações do Bond.

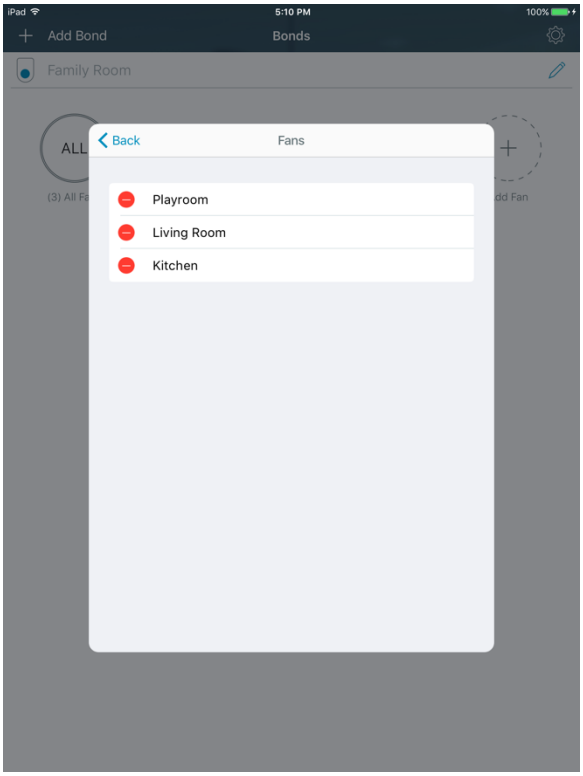

Figura 28 – Lista de dispositivos nas configurações do Bond.

Ainda na lista de Bonds (Figura 23), para cada Bond, estão disponíveis os ventiladores configurados. Os botões *All Fans* (do Inglês, todos os ventiladores),

permite que o usuário visualize os comandos cadastrados em todos os ventiladores. Já os botões de agrupamento (Figura 23), que reúnem dispositivos cadastrados em uma mesma localização, permitem que o usuário visualize os comandos presentes em ventiladores de uma mesma localização. Por fim, os botões *Add Fan* (do Inglês, adicionar ventilador), permitem a configuração de um novo ventilador para determinado Bond.

Na seção a seguir serão detalhados os passos para a configuração de um novo ventilador a ser controlado por um determinado Bond.

4.4 Configuração de Dispositivos

O primeiro passo para iniciar a configuração de um novo dispositivo, limitados aos ventiladores de teto, é pressionar o botão *Add Fan* (do Inglês, adicionar ventilador) ilustrado na Figura 25.

Em seguida o usuário será levado para uma tela (Figura 29) onde é possível inserir o código FCC do ventilador ou informar que você não possui o mesmo.

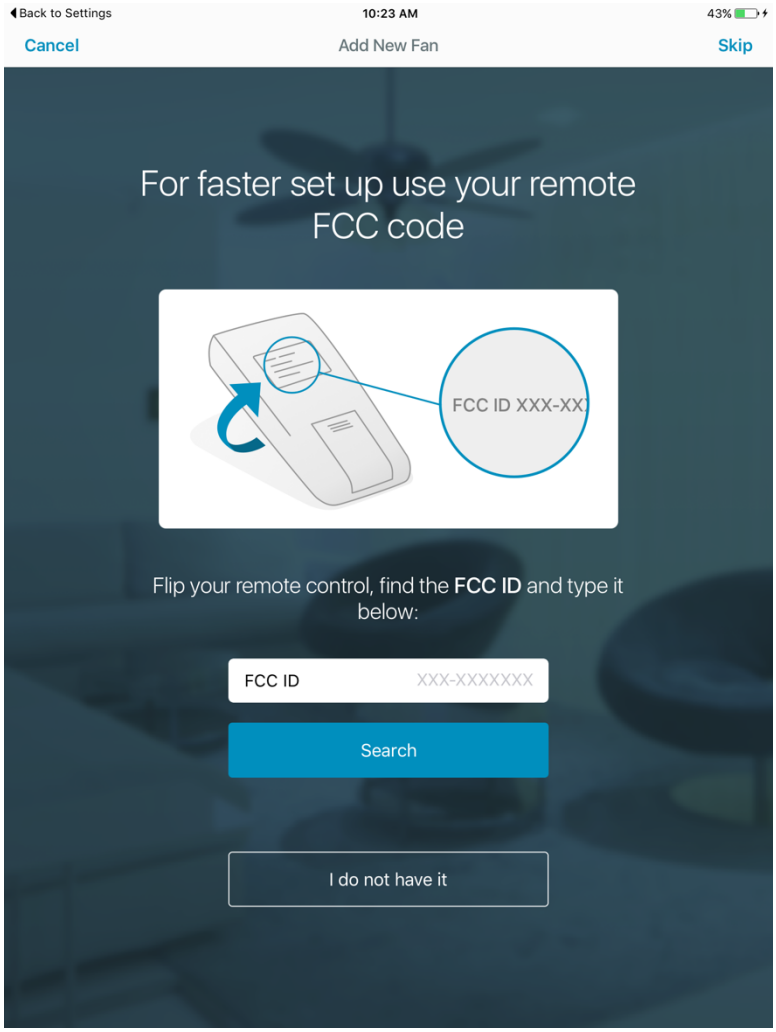

Figura 29 – Início dos passos para adicionar um novo dispositivo em um Bond.

No caso de o usuário ter o código FCC, normalmente disponível na parte de trás do controle remoto dos ventiladores, e o mesmo ser reconhecido pelo Bond, os comandos para controlar o ventilador serão configurados automaticamente, sem a necessidade de configurar cada comando a ser executado. Caso o usuário não tenha o código FCC do controle ou o mesmo não esteja presente no banco de dados do Bond, é possível configurar os comandos manualmente, como será descrito na seção 4.6. Em ambos os casos, o próximo passo é selecionar a localização do ventilador. Como a interface é bastante similar a Figura 21, a mesma será omitida.

A configuração de um ventilador só é útil ao usuário com a configuração também de seus comandos, para que o mesmo possa ser controlado. Por isso, após definir a localização, o usuário pode continuar diretamente com a configuração dos comandos para o dispositivo.

No caso de selecionar a opção para programar um comando mais tarde, pressionando o botão *Program Later* (do Inglês, programar depois), o usuário é direcionado para tela de informações daquele dispositivo (Figura 30).

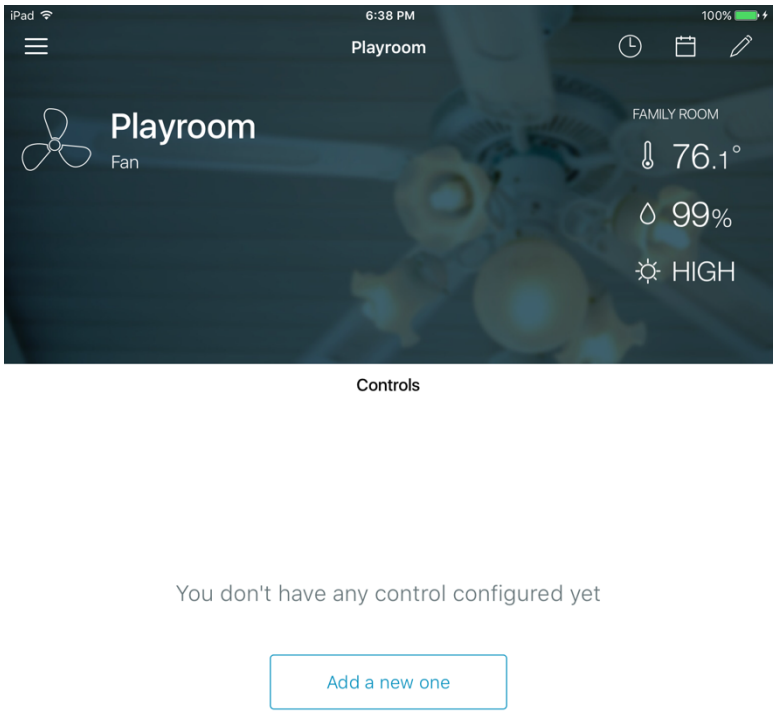

Figura 30 – Informações de um dispositivo sem comandos configurados.

A seguir serão descritos os elementos e funcionalidades presentes na tela de informações de dispositivos, que inclui a lista dos comandos cadastrados em determinado dispositivo.

4.5 Informações de Dispositivos

Na Figura 30 é possível visualizar a divisão da tela de informações em duas seções: um cabeçalho, com informações do dispositivo, Bond e funcionalidades adicionais; e a seção de comandos. Como o dispositivo não possui comandos cadastrados, a lista de comandos apresentada está vazia e uma mensagem convida o usuário a cadastrar um novo comando.

Na Figura 31 é possível visualizar um dispositivo com o comando *Speed 1* cadastrado. No cabeçalho, no canto superior esquerdo é possível pressionar um botão para retornar para a lista de Bonds. No canto superior direito, é possível visualizar três botões, um para a funcionalidade de temporizador, outro para o

agendamento de comandos e por fim, um botão para editar as configurações do dispositivo.

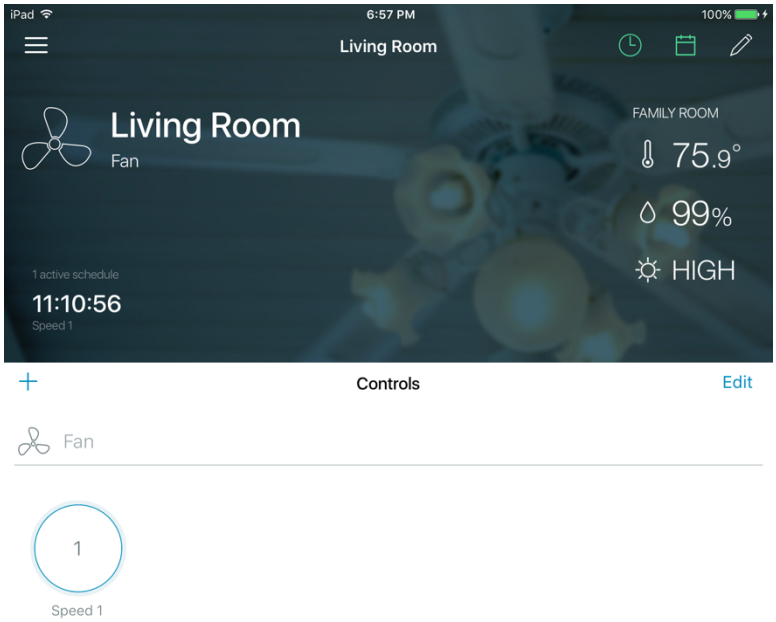

Figura 31 – Informações de um dispositivo com o comando Speed 1 configurado.

Utilizando o botão de temporizador, o ícone com formato de relógio, é possível programar um temporizador para uma determinada função (Figura 32), basta selecionar uma função e a hora que a mesma irá ser disparada. As informações do temporizador programado também podem ser visualizadas na tela de informações do dispositivo, no canto inferior esquerdo do cabeçalho (Figura 31).

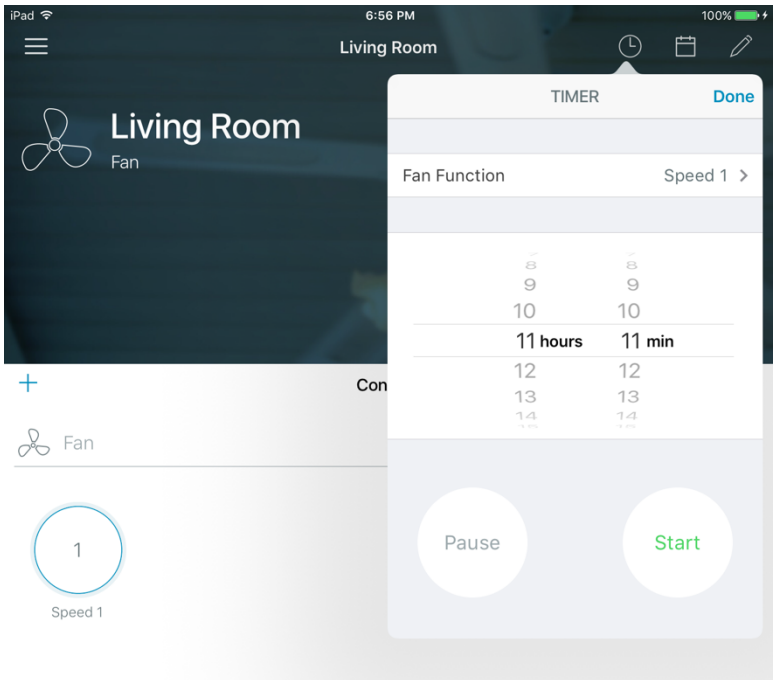

Figura 32 – Informações de um dispositivo contendo a pop-up do temporizador.

Através do ícone de calendário, é possível visualizar o calendário de agendamento de funcionalidades (Figura 33). No canto superior esquerdo é possível realizar um novo agendamento e no canto superior direito, através do botão *Done*, o usuário retorna para a tela de informações do dispositivo. Ainda, ao pressionar em um agendamento, o usuário pode editar ou deletar o mesmo. Por fim, assim como as informações do temporizador, as informações de agendamentos são apresentadas na tela de informações do dispositivo (Figura 34).

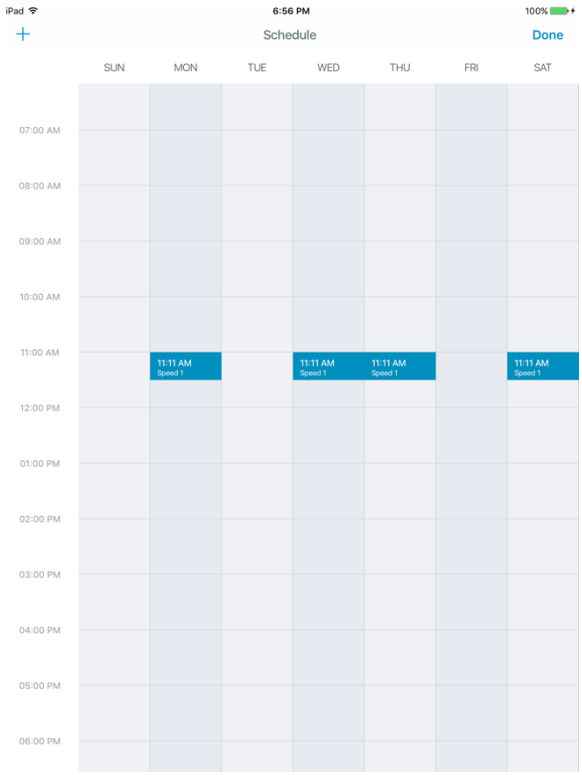

Figura 33 – Tela de agendamento de funções de um dispositivo.

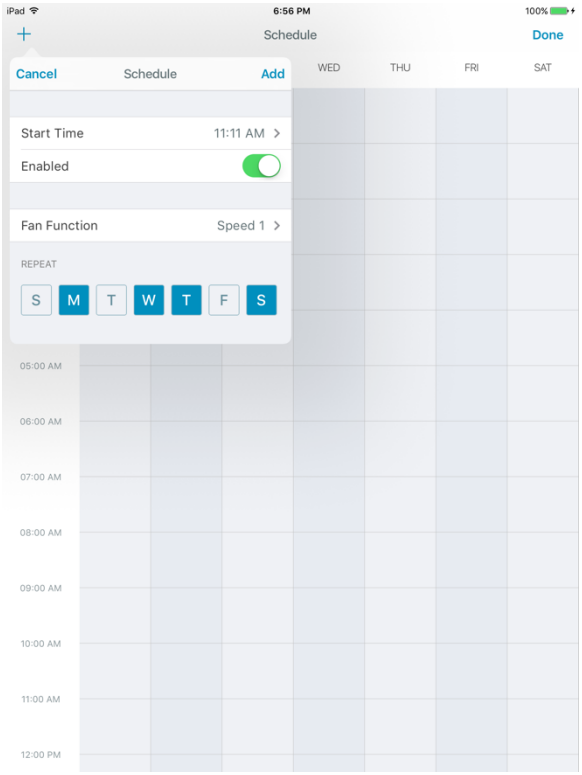

Figura 34 – Pop-up para criação ou edição de um agendamento.

Ao pressionar o botão para configuração do dispositivo, o ícone com formato de lápis, uma janela se abre, onde o usuário pode alterar a localização do dispositivo, visualizar a localização e remover o dispositivo do Bond associado (ver Figura 35).

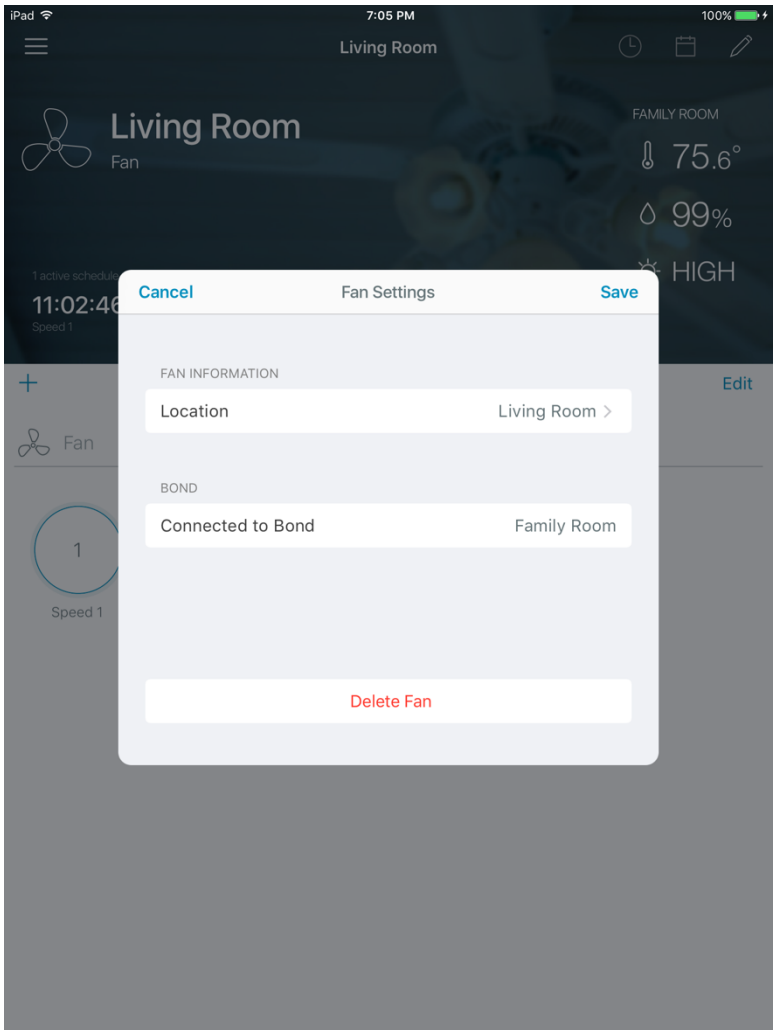

Figura 35 – Pop-up de configurações de um dispositivo.

Ainda na tela de informações (Figura 31), na parte central do cabeçalho é possível visualizar a localização do dispositivo na esquerda — Living Room (do Inglês, sala de estar) e informações do Bond, do qual o dispositivo recebe comandos, na direita, incluindo o valor dos sensores de temperatura (75.9 graus Fahrenheit), umidade (99%), luminosidade (HIGH) e sua localização — Family Room (do Inglês, sala de estar). Esses valores são apenas ilustrativos. É importante esclarecer que devido ao alcance de um sinal de rádio frequência, um Bond pode estar em uma localização e o dispositivo controlado pelo mesmo em outra, como nesse caso.

Ao pressionar um comando configurado (por exemplo, *Speed 1*), o mesmo é enviado para o Bond, que irá informar a placa de rádio frequência e essa por sua vez, enviar um sinal de rádio frequência com a função cadastrada no comando.

Por fim, através do botão com ícone de “+” é possível configurar um novo comando e ao pressionar o botão de *Edit* (do Inglês, editar) um comando pode ser removido.

Na próxima seção serão descritos os passos necessários para a programação de um novo comando para um determinado dispositivo.

4.6 Programação de Comandos

O primeiro passo para configurar um novo comando, após pressionar o botão de adicionar um comando descrito anteriormente, é escolher um botão, na lista de funções disponíveis (Figura 36), que mais se aproxime da funcionalidade no controle remoto do dispositivo a ser controlado. Os comandos configurados anteriormente parecem destacados em verde, como *Speed 1* ilustrado na mesma Figura 36. Caso um desses comandos seja selecionado, uma confirmação de reconfiguração será apresentada, como ilustrado na Figura 37.

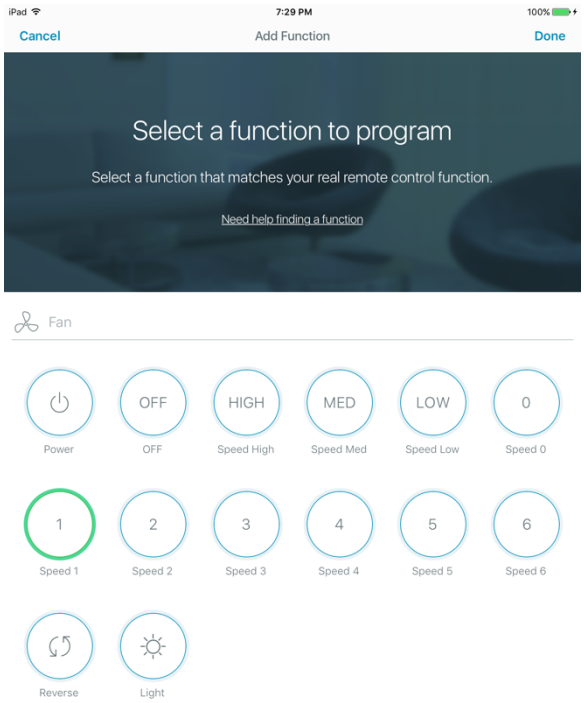

Figura 36 – Lista de funções disponíveis para serem programadas.

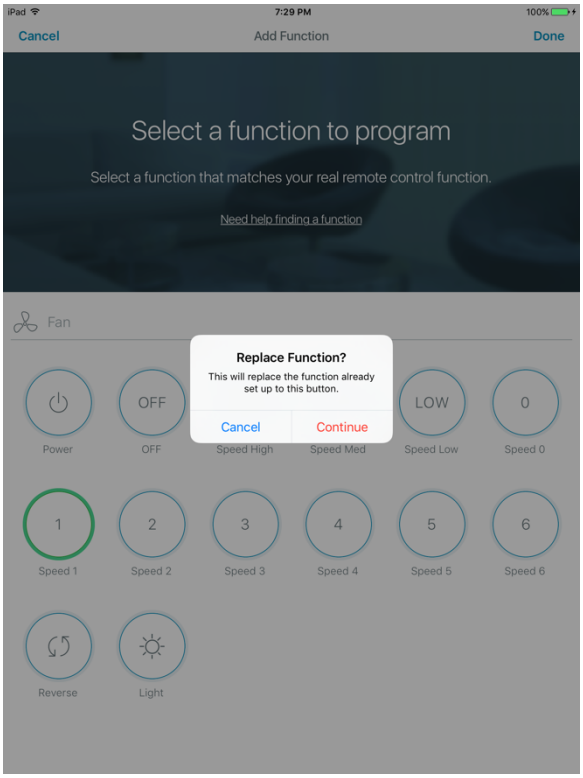

Figura 37 – Alerta de confirmação da reprogramação de uma função.

Após selecionada a função, é iniciado um passo a passo para a programação da mesma. Primeiramente é apresentada a informação de que o usuário deve aproximar o controle remoto do dispositivo, ao Bond, para que o mesmo consiga registrar o sinal utilizado para aquela função (ver Figura 38). Apesar dos sinais de rádio frequência possuírem longo alcance no contexto de uma casa, essas instruções aumentam a chance de uma boa captura do sinal configurado.

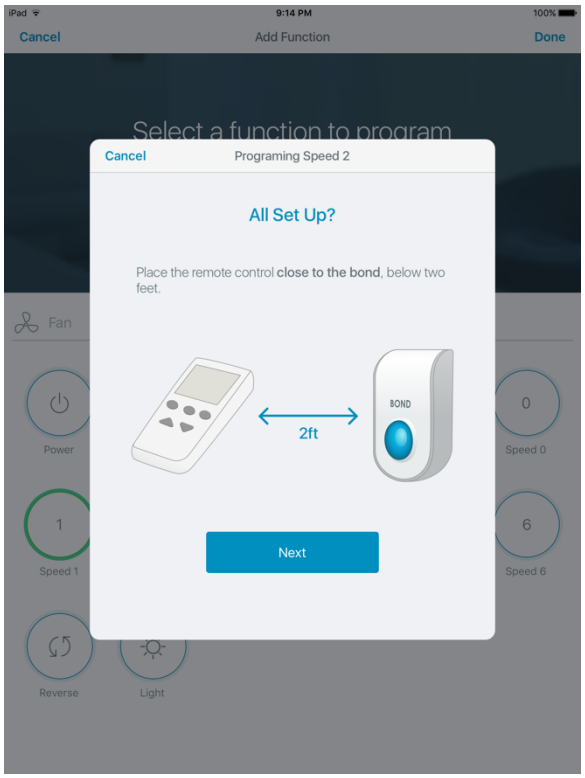

Figura 38 – Informações para iniciar a programação de um novo comando.

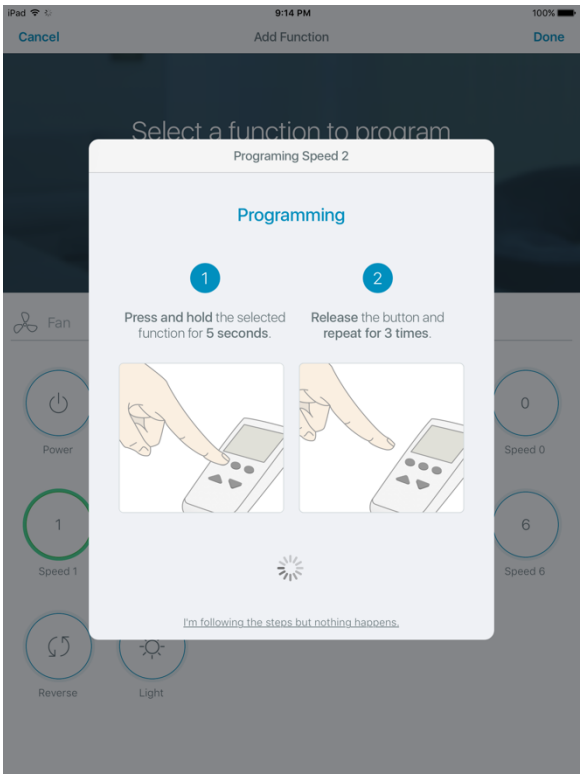

Figura 39 – Informações e programação de um novo comando.

Ao pressionar o botão de *Next* o usuário recebe novas instruções (Figura 39), solicitando que o mesmo pressione e segure o botão do controle remoto tradicional do dispositivo a ser controlado, por cinco segundos. Esse procedimento deve ser repetido três vezes, porém quando o Bond detecta e registra o sinal, o procedimento é interrompido, levando o usuário para a tela de testes (Figura 40).

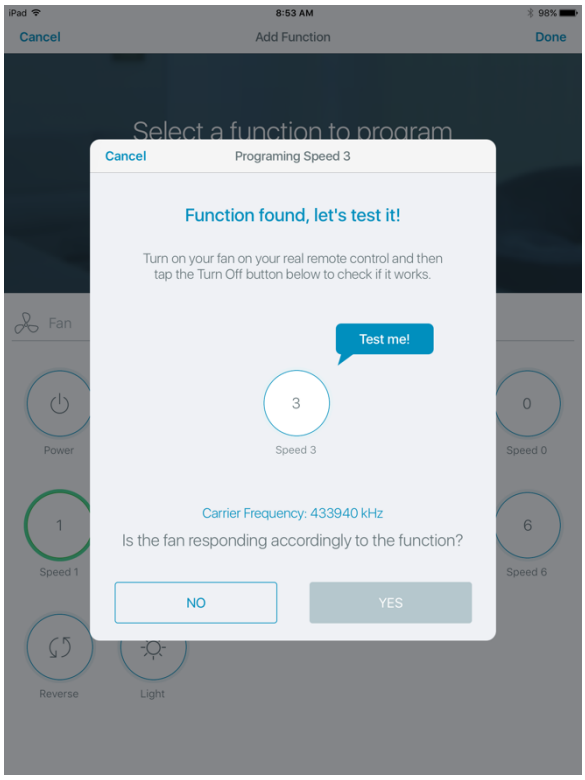

Figura 40 – Tela de testes de um comando recém programado.

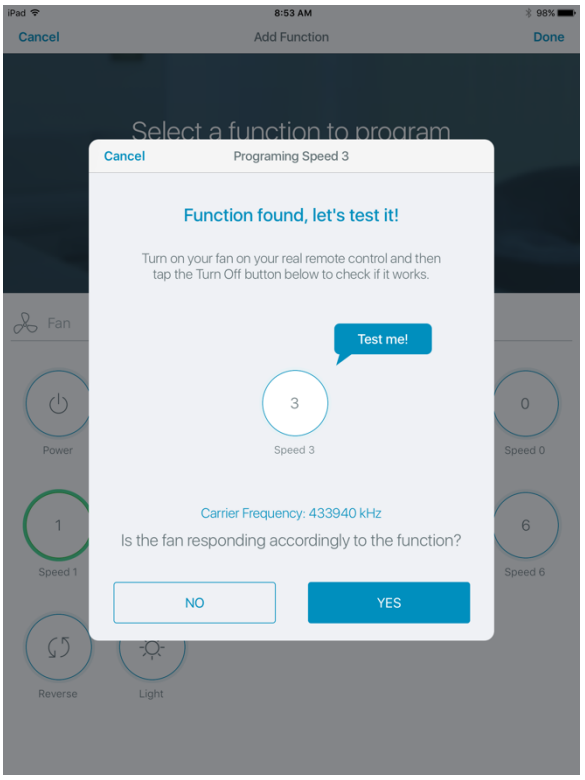

Figura 41 – Tela de testes com botões de Yes e No habilitados.

Com o cadastro de um novo sinal de rádio frequência, o usuário é solicitado a testar a mesma e verificar se o comportamento esperado foi alcançado. Por exemplo, no caso de um comando que acende a luz do ventilador de teto, o usuário irá pressionar o botão de teste no aplicativo móvel e verificar se a luz foi efetivamente ligada ou desligada, dependendo do estado inicial. Após verificar o comportamento, o usuário será habilitado a pressionar o botão *Yes* (do Inglês, sim), no caso positivo, ou *No* (do Inglês, não), em caso negativo (Figura 41).

Ao confirmar o bom comportamento do comando, realizando a função esperada, o usuário retorna à lista de comandos disponíveis (Figura 36) para serem programados e pode optar por configurar um novo comando ou retornar ao painel de informações do dispositivo (Figura 31), com os novos comandos disponíveis.

No caso de o comando não funcionar como esperado, é possível reiniciar os passos para o cadastro através do botão *No* (do Inglês, Não).

5 Implementação

A seguir serão apresentadas as principais contribuições do autor ao projeto Bond, mencionando detalhes de implementações.

Na Figura 42 é possível visualizar o sistema completo: sistema embarcado Raspberry Pi Zero junto ao módulo Wi-Fi, destacado em azul, utilizados para implementação do sistema de back-end apresentado na seção 5.3; contribuição da empresa Altus-PCB para o projeto, incluindo a placa de rádio frequência e MCU. Esses dois módulos, Raspberry Pi e MCU, estão acoplados via comunicação serial, detalhada na seção 5.3.2; Tablet/iPad rodando o aplicativo iOS abordado na seção 5.1. Detalhes da comunicação via REST API entre o aplicativo iOS e o sistema de back-end serão apresentados na seção 5.2; e, por fim, um controle remoto de ventilador de teto tradicional é ilustrado e destacado em amarelo, para exemplificar um dos vários controles remotos que serão substituídos pelo aplicativo, através do projeto Bond.

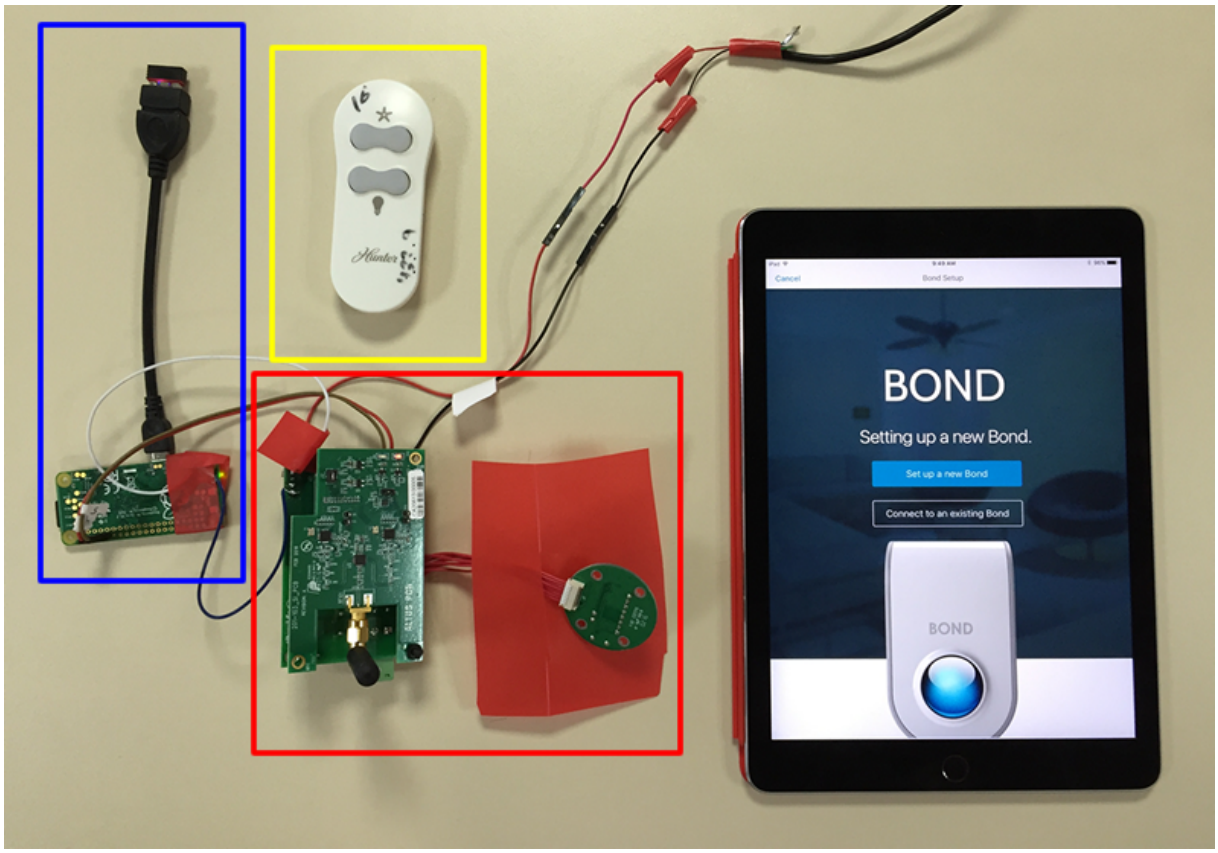

Figura 42 – Componentes da implementação do projeto Bond.

5.1 Aplicativo iOS: Bond

O Bond foi iniciado pelo desenvolvimento do aplicativo em iOS, com suporte apenas para iPads, através da codificação da UI. Apesar de existirem propostas de arquiteturas mais recentes, como VIPER [23], a arquitetura padrão utilizada para o desenvolvimento de aplicativos iOS é a MVC (*Model View Controller*) [24] e foi a escolhida pelo autor, principalmente pelo curto prazo para o desenvolvimento do projeto, sendo que o mesmo possuía experiências [22] com o uso desta arquitetura.

5.1.1 *Model*: Banco de Dados Local

O framework Core Data, nativo do iOS, foi utilizado para a persistência de dados localmente. Diferente de banco de dados relacionais, onde os dados são retornados através de um conjunto de valores, o CoreData retorna sempre um conjunto de objetos. Apesar disso, o mapeamento para o modelo relacional é direto, evitando qualquer dificuldade de compatibilidade com o sistema de back-end que utiliza um banco de dados relacional. Além de auxiliar na criação das classes dos modelos, o CoreData também possui uma interface visual para visualizar as relações definidas. Na Figura 43 é possível visualizar a estrutura visual gerada automaticamente pelo Core Data.

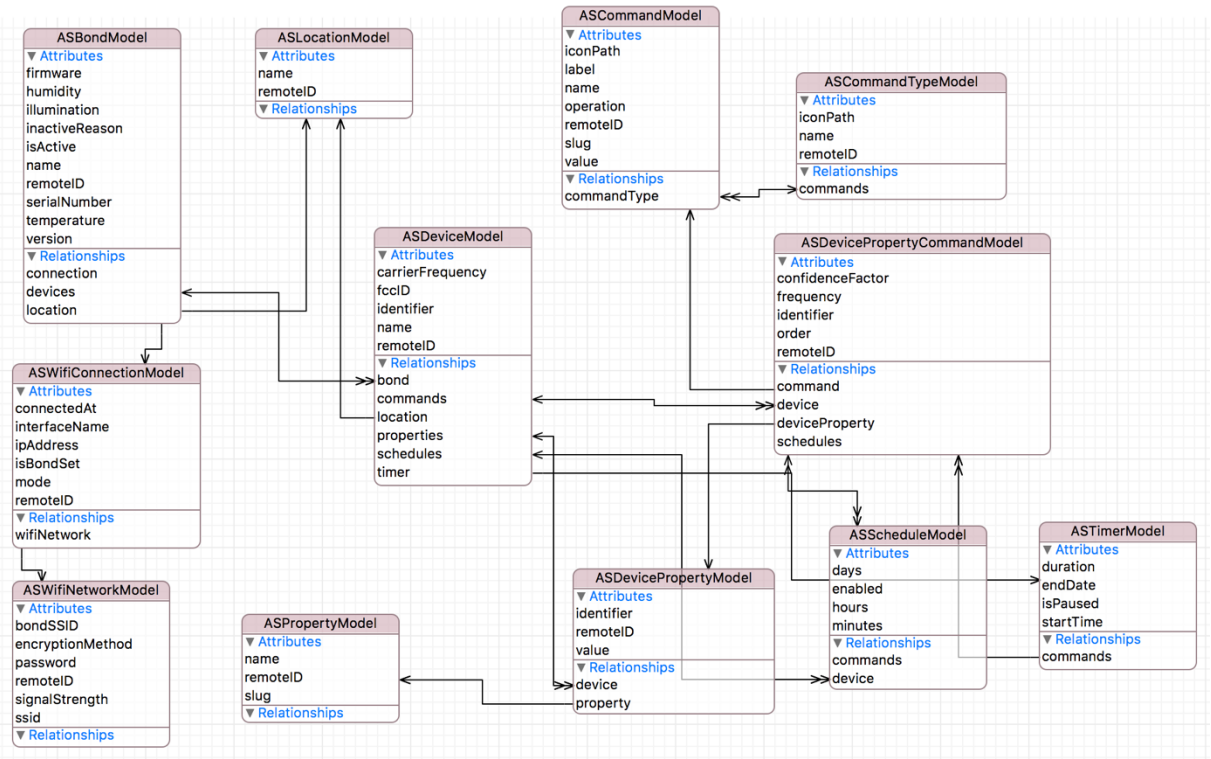

Figura 43 – Modelo de banco de dados local em CoreData.

De maneira geral, a entidade Bond armazena os valores dos sensores, temperatura, umidade e luminosidade, assim como número de fábrica (*serial number*) e valores de versão do back-end e do firmware da placa de rádio frequência que está ligada ao back-end via comunicação serial. Como cada Bond, após configurado, está conectado a rede Wi-Fi, as informações para conexão são armazenadas na entidade *wifi_connection*. As informações de uma nova conexão são mapeadas para a entidade *wifi_network*. Cada Bond possui uma localização, assim como cada dispositivo (*device*). Ainda, cada Bond possui seus dispositivos e cada dispositivo possui seus comandos, que estão conectados por propriedades. Por fim, as entidades *schedule* e *timer* são utilizadas para a temporização e agendamento de um comando, respectivamente.

É possível notar um parâmetro presente em praticamente todos as entidades: *remoteID*. Como o aplicativo pode se conectar a múltiplos Bonds, é preciso salvar o ID de cada entidade no lado do servidor, para que o aplicativo saiba mapear uma entidade remota no banco de dados local, mantendo a consistência entre ambos.

5.1.2 *View*: Interface

O desenvolvimento das interfaces foi feito via código, sem a utilização de *Storyboards* ou *Interface Builder*, o que agilizou o processo de criação, porém pode não ser a melhor solução para manutenção do aplicativo. Contudo, essa estratégia se encaixa bem quando o foco do projeto está no MVP, como é o caso deste trabalho.

Na Figura 44 é possível visualizar um trecho de código para a implementação do botão "*Setup a new Bond*" da tela inicial do aplicativo Bond, ilustrado anteriormente na Figura 14. Esse processo é feito para cada elemento de interface do aplicativo, com reuso de códigos em casos de elementos parecidos, através da definição de classes e objetos que podem ser instanciadas múltiplas vezes.

```
- (UIButton *)startButtonView {
      if (!_startButtonView) {
        _startButtonView = [[UIButton alloc] initWithFrame:CGRectMake(0, 0, 320, 55)];
        [_startButtonView setBackgroundColor:[UIColor appBlueColor]];
        [_startButtonView.layer setCornerRadius:4];
        [_startButtonView.titleLabel setFont:[UIFont fontWithName:kSFUITextLightFont
size:20]];
        [_startButtonView setTitle:NSLocalizedString(@"SETUP_NEW_BOND", nil)
forState:UIControlStateNormal];
      }
      return _startButtonView;
}
```

Figura 44 – Exemplo de Implementação da View

Nota-se na Figura 44 o trecho em destaque `NSLocalizedString`, que permite a internacionalização do aplicativo, preparando o mesmo para lidar com textos que serão exibidos em diferentes idiomas. Esse método permite traduzir os textos do aplicativo para outro idioma, sem que o tradutor tenha conhecimentos sobre o desenvolvimento de aplicativos.

5.1.3 *Controller*: Lógicas e Tratamentos de Ações

Na aplicação inteira é notável a atuação da camada de Controller, pois ela é responsável por tratar as ações realizadas na *View* e informar a camada *Model* sobre atualizações, quando necessário. Além disso, destaca-se a atuação da camada *Controller* no gerenciamento das telas do aplicativo, carregando os

componentes a cada ação do usuário. A seguir são apresentados os métodos de gerenciamento do *Controller* da tela de listagem dos Bonds (Figura 45) como exemplo de implementação.

```
- (void)viewDidLoad {
    [super viewDidLoad];
    self.bondsDataSource = [NSMutableArray new];
    for (ASBondModel *bond in [ASBondModel MR_findAllSortedBy:@"location.name"
ascending:YES]) {
        [self.bondsDataSource addObject:bond];
    }
    self.bondsServiceDiscovery = [NSMutableDictionary new];
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(checkCurrentlyNetwork)

name:UIApplicationWillEnterForegroundNotification
                                               object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self
                                             selector:@selector(stopBondsListPolling)

name:UIApplicationDidEnterBackgroundNotification
                                               object:nil];
    [self setupView];
}

- (void)viewWillAppear:(BOOL)animated {
    [super viewWillAppear:animated];
    [self startGoogleAnalytics];
    [self setupNavigationBar];
    [self checkCurrentlyNetwork];
}

- (void)viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];
    [self stopBondsListPolling];
}

- (void)viewDidDisappear:(BOOL)animated {
    [super viewDidDisappear:animated];
    [self.bondsDashboardView.collectionHotspotView setHidden:YES];
    [self.bondsDashboardView.collectionView reloadData];
}
```

Figura 45 – Métodos padrões da camada Controller

É possível notar os métodos que são chamados pelo sistema após carregar a *View* (*viewDidLoad*), quando a *View* irá aparecer (*viewWillAppear*), quando a *View* irá desaparecer (*viewWillDisappear*) para dar espaço para a próxima tela e por fim, quando a *View* desapareceu (*viewDidDisappear*).

Após carregar a *View*, o Controller busca do *Model* os Bonds cadastrados ordenados alfabeticamente por sua localização através do método `[ASBondModel MR_findAllSortedBy:@"location.name" ascending:YES])` e adiciona os mesmos em uma

estrutura que será utilizada pela *View* para mostrar os mesmos na tela. Além disso, o Controller ainda define funções que serão chamadas no caso de o usuário fechar o aplicativo e retornar para o mesmo.

Os demais métodos também atuam entre a *View* e *Model* com o objetivo de manter a aplicação em funcionamento, tratando as ações do usuário na *View* e solicitando os dados ao *Model* sempre que necessário. Praticamente todas as telas do aplicativo possuem um Controller e um View e esses métodos estão presentes em todas elas, chamando diferentes métodos de acordo com a necessidade da aplicação.

5.2 Comunicação entre Aplicativo e Back-end

A comunicação entre o aplicativo e o sistema embarcado que hospeda o back-end é essencial para o funcionamento da aplicação como um todo, pois é através dessa comunicação que os comandos são enviados para controlar um dispositivo, como um ventilador.

O aplicativo se comunica com o back-end através de uma REST API via Wi-Fi, porém antes de começar a comunicação e troca de dados, o aplicativo precisa encontrar os dispositivos Bond na rede Wi-Fi, através do Bonjour.

5.2.1 Descoberta Automática via Bonjour

Para que o aplicativo iOS consiga encontrar os Bonds presentes na mesma Wi-Fi, cada Bond publica um serviço do tipo *_bond* através do protocolo Bonjour — nesse caso, como o sistema embarcado roda o sistema operacional Linux, o nome dado ao protocolo é Avahi. No serviço, além de informações como IP e porta, o Bond fornece o seu identificador único.

Ao iniciar a procura, o aplicativo busca por serviços do tipo *_bond* disponíveis na rede. Quando encontra um serviço, ele tenta resolver o mesmo, ou seja, verifica se aquele serviço é alcançável. Ao alcançar um serviço o protocolo notifica e o desenvolvedor pode optar por aguardar pela descoberta de mais serviços, ou parar a procura. Como o aplicativo iOS quer atuar em diferentes Bonds, é preciso permitir que mais serviços — que representam mais Bonds — sejam encontrados. Uma estratégia para descobrir mais Bonds, sem tornar a procura uma espera

indeterminada, é definir um temporizador que limita o tempo máximo de procura. Após atingir esse tempo, o aplicativo considera que encontrou todos os Bonds disponíveis na rede Wi-Fi e retorna uma lista com o endereço e identificador de todos.

Na Figura 46 é apresentado o principal método de descoberta automática de Bonds via Bonjour, onde é possível notar a utilização de um temporizador, assim como o tratamento da espera por mais serviços.

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)serviceBrowser
didFindService:(NSNetService *)service moreComing:(BOOL)moreComing {
    // Stop timer to handle services
    if ([self.browsingTimer isValid]) {
        [self.browsingTimer invalidate];
    }
    // Update Services
    [self.services addObject:service];
    // Resolve services
    if(!moreComing) {
        NSLog(@"Number of services found: %lu", (unsigned long)[self.services count]);
        // Fetch Services
        for (NSNetService *service in self.services) {
            // Resolve Service
            [service setDelegate:self];
            [service resolveWithTimeout:10.0];
        }
    }
}
```

Figura 46 – Método para descoberta automática de Bonds via Bonjour.

Com posse da lista de endereços e identificadores, o aplicativo executa um chamado para cada Bond, verificando se os mesmos estão disponíveis através da REST API. Aqueles que retornam uma resposta positiva, são então adicionados a aplicação e tratados como Bonds operáveis pelo aplicativo.

5.2.2 REST API

Com posse das informações de endereço de cada Bond, o aplicativo consegue enviar requisições para cada um deles através dos chamados *endpoints*, como apresentado na seção 3.2.1.

No aplicativo iOS são criadas classes de API para cada entidade, com métodos para cada *endpoint*, que possuem funções de *Callback* para tratar as respostas, seja ela uma resposta útil ou um erro. Na Figura 47 é apresentada um

exemplo de método e suas funções de *Callback*, com o objetivo de ler os sensores do Bond. A função `handleHttpRequestSuccess` trata as respostas úteis e a função `handleHttpRequestFailure` lida com erros ou respostas de mal funcionamento.

Destaca-se a utilização do endereço do Bond para formar o endereço do *endpoint (*`bond.connection.ipAddress`*)*, visto que o mesmo endpoint está presente em todos os Bonds e é preciso passar para aplicação exatamente qual Bond está sendo requisitado.

```
+ (void)fetchBondSensorsFromBond:(ASBondModel *)bond
                       onSuccess:(void (^)(ASBondModel *))successCallback
                       onFailure:(void (^)(NSError *))failureCallback {

    void (^handleHttpRequestSuccess)(AFHTTPRequestOperation *, id) =
^(AFHTTPRequestOperation *operation, id responseObject) {
        NSDictionary *jsonResponse = operation.responseObject;
        bond.temperature = [jsonResponse objectForKey:@"temperature"];
        bond.illumination = [jsonResponse objectForKey:@"light"];
        bond.humidity = [jsonResponse objectForKey:@"humidity"];
        successCallback(bond);
    };

    void (^handleHttpRequestFailure)(AFHTTPRequestOperation *, NSError *) =
^(AFHTTPRequestOperation *operation, NSError *error) {
        failureCallback(error);
    };

    [[ASBondApi makeHttpRequestOperationManager] GET:[ASBondApi
bondUrlForBond:bond.connection.ipAddress]
                                          parameters:nil
                                             success:handleHttpRequestSuccess
                                             failure:handleHttpRequestFailure];
}
```

Figura 47 – Método para coletar informações de sensores de um Bond.

Para o exemplo em questão, após receber os valores dos sensores do back-end hospedado no sistema embarcado, o aplicativo iOS armazena os valores no banco de dados local e retorna as informações atualizadas para a aplicação (`successCallback(bond);`), que irá mostrar os novos valores para o usuário.

5.3 Back-end em Sistema Embarcado

O sistema de back-end implementa os modelos que serão salvos no banco de dados remoto e acessados pelos aplicativos iOS, mantendo todas as informações de cada Bond consigo. Além disso, o back-end é responsável por efetuar a comunicação serial com a placa de rádio frequência, que contém uma MCU. A

Figura 48 ilustra a configuração e comunicações da integração, considerando apenas um aplicativo iOS, um Bond e um dispositivo a ser controlado, nesse caso um ventilador.

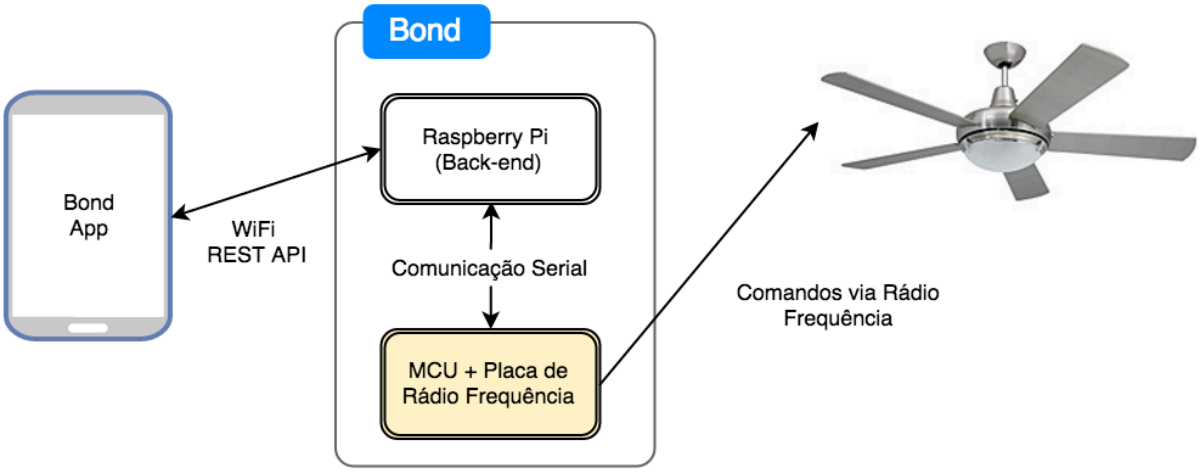

Figura 48 – Integração entre componentes do projeto Bond.

5.3.1 Banco de Dados Remoto

O aplicativo iOS enxerga múltiplos Bonds e o seu banco de dados local acomoda essa configuração, diferente do banco de dados remoto, hospedado em cada Bond, que salva as informações de apenas um Bond. Apesar disso, os bancos de dados locais e remoto são similares e o modelo do banco de dados remoto, relacional, pode ser visualizado na Figura 49.

As principais diferenças estão na não necessidade de entidades relacionadas as conexões, visto que o Bond, após configurado, permanece conectado em apenas uma rede Wi-Fi e não visualiza outros Bonds.

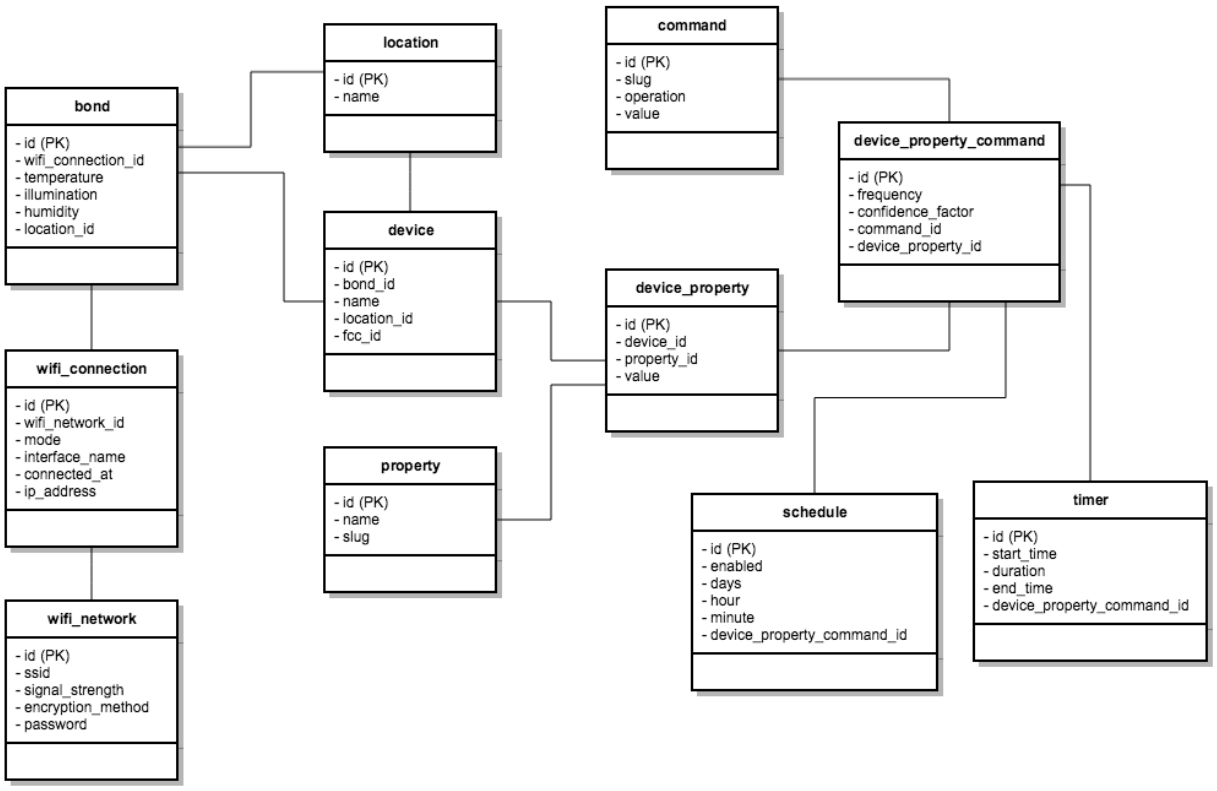

Figura 49 - Modelo relacional.

Na Figura 50 é possível visualizar a definição do modelo para a entidade Bond, utilizando a linguagem Javascript e o framework Node.js. Os demais modelos seguem o mesmo padrão e serão omitidos.

```
'use strict';
module.exports = function (sequelize, DataTypes) {
    var Bond = sequelize.define('Bond', {}, {
        classMethods: {
            associate: function (models) {
                Bond.belongsTo(models.Location, {as: 'location', foreignKey: 'locationId'});
                Bond.hasMany(models.Device, {as: 'devices', foreignKey: 'bondId'});
            }
        }
    });
    return Bond;
};
```

Figura 50 – Definição de modelo para entidade Bond.

5.3.2 Comunicação Serial

Como mencionado anteriormente, a comunicação serial desenvolvida por ambas as empresas, pela Cheesecake Labs na placa Raspberry Pi Zero e pela Altus PCB na MCU da placa de rádio frequência, acopla a parte lógica do back-end com os sensores de temperatura, umidade e luminosidade, além da placa que envia e recebe sinais de rádio frequência.

Uma vez definidos a interface de comunicação entre os dois lados e a taxa de transmissão da troca de informações, as duas placas passaram a se comunicar. As mensagens (Figura 51, Figura 52, Figura 53 e Figura 54) ilustram os principais comandos trocados pelo back-end (Node Server) e a placa de rádio frequência (RF Board).

Node Server: ">ReadSensors,requestId=1\n"
RF Board Response: ">ReadSensors,requestId=1,success=true,temp=21.5,humidity=80.3,light=268\n"

Figura 51 – Comando para ler a temperatura, umidade e luminosidade.

Node Server: ">RegisterNewFunction,requestId=2,fanId=B1,fcc=true,carrierFreq=303150,functionId=A8,appliance=fan,functionType=dimmer,remoteScheme=AB43Df56\n"
RF Board Response: ">RegisterNewFunction,functionId=A8,requestId=2,fanId=B1,acquisitionStatus=inProgress,success=true\n"

Figura 52 – Comando para programar uma função/comando.

Node Server: ">ExecuteFunction,requestId=3,functionId=AE\n"
RF Board Response: ">ExecuteFunction,requestId=3,success=true\n"

Figura 53 – Comando para executar uma função.

Node Server:">ProgramStatus,requestId=5,functionId=1C\n"
RF Board Response:">ProgramStatus,requestId=5,acquisitionStatus=done,signalStrength=77,functionId=3B,confidenceFactor=96,remoteCommand=AADFFFA9343Da,carrierFreq=303.15M, functionRecorded=A3442DC4F,remoteScheme=33F\n"

Figura 54 – Comando para verificar se o sinal programado pode ser testado.

A comunicação serial representa um gargalo para o sistema, principalmente pelo fato de a placa de rádio frequência não conseguir se comunicar com um *baud*

rate maior de 9600 bits por segundo. Essa demora para receber os comandos pode acarretar em uma experiência negativa para o usuário, que irá pressionar um botão no aplicativo móvel para controlar o dispositivo, como um ventilador, e poderá ter que esperar por segundos para notar o efeito. Certamente uma solução com placa única para o back-end e o transmissor de sinais de rádio frequência será projetada para o futuro, eliminando a necessidade de uma comunicação serial entre ambas e evitando qualquer imagem negativa que o usuário possa ter em relação ao produto.

Por fim, mesmo não sendo uma solução sustentável para um produto em larga escala, a comunicação serial foi a última peça para a concretização do projeto, na versão MVP. Com ela, todo o circuito de comunicação entre o aplicativo móvel e um dispositivo a ser controlado, foi fechado.

6 Conclusões e Perspectivas

O objetivo do projeto Bond versão MVP era permitir que usuários pudessem cadastrar funções de controles remotos de ventiladores de teto em seu Bond, utilizando o mesmo como solução integrada para o controle de todos os ventiladores de sua residência. Apesar da complexidade do produto, um dos principais requisitos era o tempo de desenvolvimento, que deveria ser curto para a validação da proposta em apresentações com investidores nos Estados Unidos. Por isso, alguns requisitos de custos de produção foram ignorados e o trabalho foi desenvolvido em um balanço entre complexidade, custos e rapidez.

Vale comentar que o desafio do projeto não foi limitado a esse trabalho, mas também envolveu o desenvolvimento da placa de rádio frequência, pela empresa Altus-PCB, que transmite, recebe e registra sinais de rádio frequência, com diferentes métodos de codificação.

Com a versão atual, o usuário consegue cadastrar controles remotos de ventiladores de teto, assim como comandar os mesmos. Considerando a situação onde o usuário envia um comando para ligar a luz de um ventilador de teto, toda a comunicação — incluindo a requisição pelo aplicativo iOS ao back-end em sistema embarcado, comunicação com a placa de rádio frequência e envio do sinal de comando da mesma para o ventilador — leva em torno de dois segundos. Apesar de não ser comparável com o tempo de comunicação utilizando o controle remoto tradicional, é um resultado satisfatório, considerando alguns gargalos bem críticos, como a comunicação serial.

Ao fim do período de desenvolvimento, de aproximadamente 4 meses, o produto foi apresentado pelo cliente, fundador da empresa Altus-PCB, em cinco apresentações para investidores. O resultado foi considerado excelente e principalmente, permitiu a validação do produto, com interesse de empresas de grande porte do setor, como Home Depot e Lowes.

É possível concluir que as contribuições do autor ao projeto tiveram grande relevância, tendo em vista que elas permitiram a concretização do objetivo do MVP: comandar através de um aplicativo móvel para iOS/iPad um dispositivo controlado

via sinais de rádio frequência. Em breve, pretende-se implementar a versão para iPhone e também dar suporte ao sistema Android nas mais variadas marcas de celulares.

O desenvolvimento de um produto com foco no MVP, do início ao fim, foi de grande relevância para manter as atividades alinhadas ao escopo definido, além de ter sido uma abordagem de desenvolvimento nova para o autor. Com essa metodologia, o produto pôde ser validado no mercado, antes de receber um aporte maior de investimentos para ir para produção.

Durante o processo de desenvolvimento novas ideias foram listadas para as versões futuras, assim como mudanças que serão necessárias para a produção em escala. Primeiramente será necessário substituir o sistema embarcado Raspberry Pi Zero por uma placa de mais baixo nível, como uma MCU, com um módulo Wi-Fi integrado, reduzindo significativamente os custos de produção em escala e por consequência os custos do produto final.

Por fim, este trabalho foi de grande importância para o crescimento pessoal e profissional do autor, tanto em função dos conhecimentos específicos adquiridos em desenvolvimento de software quanto da vivência do trabalho em equipe e da responsabilidade de concretizar o projeto em tempo hábil para o cliente atender as reuniões de apresentação do mesmo para investidores.

Bibliografia:

[1] TecnoBlog. De olho na “internet das coisas”, Google compra Nest por US$ 3,2 bilhões. Disponível em: https://tecnoblog.net/148904/google-compra-nest/. Acesso em: 10/05/2016.

[2] The Verge. Apple announces Home app for iOS 10. Disponível em: http://www.theverge.com/2016/6/13/11923868/apple-home-app-ios-10-homekit-smart-home. Acesso em: 11/05/2016.

[3] ACEEE. DOE just issued a proposed rule for ceiling fans and a final rule for ceiling fan light kits. Disponível em http://aceee.org/blog/2015/12/doe-just-issued-proposed-rule-ceiling. Acesso em: 11/05/2016.

[4] Resultados Digitais. Como a Cheesecake Labs obteve um ROI de 22x qualificando seus Leads. Disponível em http://blog.rdstation.com.br/estudos-de-caso/como-a-cheesecake-labs-obteve-roi-22x/. Acesso em 16/06/2016.

[5] Slack. Ferramenta para comunicação de times de trabalho. Disponível em: https://slack.com.

[6] Google Hangout. Plataforma de mensagens instantâneas e chat de vídeo. Disponível em: https://hangouts.google.com.

[7] GitHub. Portal que oferece serviços relacionados ao git, um sistema de controle de versão distribuído. Disponível em: https://github.com.

[8] Pivotal Tracker. Portal que oferece serviços para gerenciamento de tarefas. Disponível em: https://www.pivotaltracker.com.

[9] Apple Inc. *iOS Technology Overview*. Disponível em: https://developer.apple.com/library/ios/documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html. Acesso em: 18/06/2016.

[10] Apple Inc. About Objective-C. Disponível em: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html. Acesso em: 31/08/2016.

[11] Apple Inc. *Start Developing iOS Apps (Swift).* Disponível em: https://developer.apple.com/library/ios/referencelibrary/GettingStarted/DevelopiOSAppsSwift/. Acesso em: 16/06/2016.

[12] *CocoaPods.* Disponível em: https://cocoapods.org/. Acesso em: 18/06/2016.

[13] Roy T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. 2000. Disponível em: http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm. Acesso em: 18/06/2016.

[14] Apiary. API Blueprint Tutorial. Disponível em: https://help.apiary.io/api_101/api_blueprint_tutorial/. Acesso em: 20/05/2016.

[15] iMasters. O que exatamente é o Node.js? Disponível em: http://imasters.com.br/artigo/22016/javascript/o-que-exatamente-e-o-nodejs/. Acesso em 19/06/2016.

[16] WebOfThings. Node.js for Embedded Systems. Disponível em http://webofthings.org/2016/06/18/node-js-for-embedded-systems/. Acesso em 20/06/2016.

[17] TechBlog Netflix. Node.js in Flames. Disponível em http://techblog.netflix.com/2014/11/nodejs-in-flames.html. Acesso em 20/06/2016.

[18] Tableless. Consumo eficiente de recursos computacionais de servidores de aplicação web com Node.js. Disponível em: http://tableless.com.br/consumo-eficiente-de-recursos-computacionais-de-servidores-de-aplicacao-web-com-node-js/. Acesso em: 20/06/2016.

[19] JavaScript para World Wide Web, Tradução 3ª. Edição, Visual QuickStart Guide, Autores: Tom Negrino e Dori Smith, 1999, Editora Campus. ISBN 85-352-0622-1.

[20] Apple Inc. Bonjour Overview. Disponível em: https://developer.apple.com/library/mac/documentation/Cocoa/Conceptual/NetServices/Introduction.html. Acesso em: 23/06/2016.

[21] Adafruit. Bonjour (Zeroconf) Networking for Windows and Linux. Disponível em: https://learn.adafruit.com/bonjour-zeroconf-networking-for-windows-and-linux/overview. Acesso em: 23/06/2016.

[22] Processamento de Sinais em Aplicativo de Aparelho Auditivo. Relatório submetido à Universidade Federal de Santa Catarina como requisito para aprovação na disciplina DAS 5501: Estágio em Controle e Automação Industrial. Autor: Antonio Adalberto Duarte Júnior, 2013.

[23] Objc.io. Architecting iOS Apps with VIPER. Disponível em: < https://www.objc.io/issues/13-architecture/viper/>. Acesso em: 24/06/2016.

[24] Apple Inc. Model-View-Controller. Disponível em: https://developer.apple.com/library/mac/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html. Acesso em: 14/04/2016.