

**Universidade do Minho**

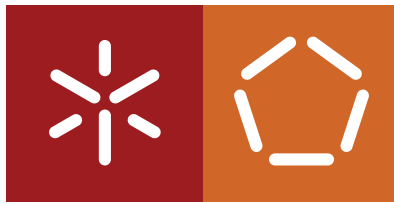
Escola de Engenharia

Departamento de Informática

Leonel da Cruz Gonçalves

**Aplicações híbridas para Smart Homes**





**Universidade do Minho**  
Escola de Engenharia  
Departamento de Informática

Leonel da Cruz Gonçalves

## **Aplicações híbridas para Smart Homes**

Dissertação de Mestrado  
Mestrado em Engenharia Informática

Trabalho efetuado sob a orientação de  
**António Nestor Ribeiro**

---

## DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

---

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

LICENÇA CONCEDIDA AOS UTILIZADORES DESTE TRABALHO:



**CC BY**

<https://creativecommons.org/licenses/by/4.0/>

---

## AGRADECIMENTOS

---

Terminada esta etapa importante da minha vida, gostaria de agradecer a todos que de alguma forma contribuíram para a realização desta dissertação, às quais transmito os meus mais sinceros agradecimentos.

Primeiramente ao meu orientador, Prof. Nestor Ribeiro, pela confiança e apoio prestado durante esta etapa. Sem a sua supervisão e as suas opiniões críticas, a realização deste trabalho não seria possível. Um muito obrigado.

Um profundo e sincero obrigado à minha família por toda a ajuda e educação que me proporcionaram ao longo da minha vida e especialmente durante esta fase. Aos meus amigos, por me acompanharem no decorrer do meu percurso académico, desde as tardes de estudo às noites de diversão, e que fizeram de mim a pessoa que sou hoje. Um agradecimento especial à Ana, Daniel, Tiago, Hugo, Ricardo e Dany pela amizade e por estarem sempre comigo, nos bons e nos maus momentos. A todos os que não mencionei, mas que de alguma forma contribuíram para que tudo isto tenha valido a pena, o meu maior obrigado.

---

## DECLARAÇÃO DE INTEGRIDADE

---

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Universidade do Minho, 7 de Dezembro de 2020

Nome Completo:

Leonel da Cruz Gonçalves

Assinatura:

Leonel da Cruz Gonçalves

---

## ABSTRACT

---

Internet of Things concept has achieved a huge popularity in the last years due to the quality of life its use is able to provide its users. Among the several areas where this concept is applied, Smart Homes are one of the most popular, which are increasingly a reality in our daily lives.

In a smart home, in addition to the IoT devices and a Hub that connects them to each other, it's indispensable the use of an application that allows the management of their equipment, regardless of the location around the house. In this context, whenever a company decides to invest in this business sector, among which several decisions are made in the initial phase of the project, stands out the choice of the type of application to be developed to control the different equipment. This choice may prove difficult due to the little information available regarding the behavior of application types changes in this context.

The goal of this dissertation is to study one of the existing types of applications, hybrid applications, and to bridge the analysis results in the development of one that allows the management of a smart home. This application will allow the analysis of the behavior of hybrid applications in the context of Smart Homes, providing study tools for the evolution of this business area.

**KEYWORDS** Hybrid Applications, Internet of Things, Smart Homes

---

## RESUMO

---

O conceito de Internet of Things alcançou uma enorme popularidade ao longo dos últimos anos graças à qualidade de vida que a sua utilização consegue proporcionar aos seus utilizadores. Entre as diversas áreas onde este conceito é aplicado, destacam-se as Smart Homes, ou casas inteligentes, que são cada vez mais uma realidade no nosso quotidiano.

Numa casa inteligente, para além dos dispositivos de IoT e de um Hub que os conecta entre si, é indispensável a existência de uma aplicação que possibilite ao utilizador a gestão integrada dos seus equipamentos, independentemente da localização destes pela casa. Nesse sentido, sempre que uma empresa decide investir neste setor de negócio, entre as diversas decisões que são tomadas na fase inicial do projeto, destaca-se a escolha do tipo de aplicação a desenvolver para controlar os diferentes equipamentos. Esta escolha pode revelar-se difícil devido à pouca informação existente no que diz respeito ao comportamento das respetivas tipologias neste contexto.

O objetivo desta dissertação passa por estudar uma das tipologias de aplicações existentes, as híbridas, e colmatar os resultados da análise no desenvolvimento de uma aplicação que possibilite a gestão de uma casa inteligente. A sua conceção irá permitir a análise do comportamento das aplicações híbridas no contexto das Smart Homes, fornecendo assim ferramentas de estudo para o desenvolvimento deste setor.

**PALAVRAS-CHAVE**    Aplicações Híbridas, Internet of Things, Smart Homes



---

## CONTEÚDO

---

|  |    |
|--|----|
| <b>Índice</b> .....  | i  |
| <b>1 INTRODUÇÃO</b> .....  | 1  |
| 1.1 Enquadramento .....  | 1  |
| 1.2 Motivação .....  | 2  |
| 1.3 Objetivos .....  | 2  |
| 1.4 Estrutura do Documento .....   | 3  |
| <b>2 ESTADO DA ARTE</b> .....  | 5  |
| 2.1 Internet of Things .....   | 5  |
| 2.1.1 Definição .....  | 5  |
| 2.1.2 Objetivos da IoT .....   | 6  |
| 2.1.3 Crescimento da IoT .....   | 6  |
| 2.2 Smart Home .....   | 7  |
| 2.2.1 Arquitetura de Smart Homes .....   | 8  |
| 2.2.2 Utilizadores .....   | 10 |
| 2.2.3 Tecnologias .....  | 10 |
| 2.3 Aplicações móveis .....  | 17 |
| 2.3.1 Aplicações nativas .....   | 18 |
| 2.3.2 Aplicações Web .....   | 19 |
| 2.3.3 Aplicações Híbridas .....  | 19 |
| 2.3.4 Análise comparativa entre o desenvolvimento de aplicações móveis Nativas, Web e Híbridas ..... | 20 |
| 2.4 Definição das tecnologias Híbridas .....   | 23 |
| 2.4.1 React Native .....   | 23 |
| 2.4.2 Ionic framework .....  | 24 |
| 2.4.3 Apache Cordova .....   | 25 |
| 2.4.4 Angular .....  | 25 |
| 2.5 Arquitetura do middleware desenvolvido .....   | 26 |

|          |   |           |
|----------|---|-----------|
| <b>3</b> | <b>ANÁLISE E CONCEÇÃO</b>                 | <b>28</b> |
| 3.1      | Análise de requisitos                     | 28        |
| 3.1.1    | Requisitos funcionais                     | 29        |
| 3.1.2    | Requisitos não funcionais                 | 32        |
| 3.2      | Conceção                                  | 37        |
| 3.2.1    | Modelo de Domínio                         | 37        |
| 3.2.2    | Casos de uso                              | 39        |
| 3.2.3    | Diagramas de classes                      | 42        |
| <b>4</b> | <b>IMPLEMENTAÇÃO, TESTES E DEPLOYMENT</b> | <b>46</b> |
| 4.1      | Ferramentas e Tecnologias                 | 46        |
| 4.1.1    | Seleção da Framework Híbrida              | 46        |
| 4.1.2    | Node.js                                   | 47        |
| 4.1.3    | OpenWeatherMap                            | 48        |
| 4.1.4    | OneSignal                                 | 49        |
| 4.2      | API do middleware                         | 50        |
| 4.2.1    | Extensibilidade do Middleware             | 50        |
| 4.2.2    | Utilização da API                         | 53        |
| 4.3      | Descrição da Implementação                | 54        |
| 4.3.1    | Estrutura da Aplicação                    | 54        |
| 4.3.2    | Consumo de APIs nativas                   | 55        |
| 4.3.3    | Componentes Ionic                         | 57        |
| 4.3.4    | Integração com Angular                    | 57        |
| 4.3.5    | Gráficos                                  | 59        |
| 4.3.6    | Armazenamento local de informação         | 60        |
| 4.3.7    | Assistência Multi-Idioma                  | 61        |
| 4.3.8    | Notificações Push                         | 62        |
| 4.3.9    | Segurança                                 | 64        |
| 4.4      | Testes                                    | 65        |
| 4.5      | Deployment                                | 67        |
| 4.5.1    | Repositório público                       | 67        |
| 4.5.2    | Heroku                                    | 68        |
| 4.5.3    | APK                                       | 68        |
| <b>5</b> | <b>RESULTADO FINAL</b>                    | <b>70</b> |

|          |   |           |
|----------|---|-----------|
| 5.1      | Ambiente de execução .....                                    | 70        |
| 5.2      | Demonstração.....   | 70        |
| <b>6</b> | <b>ANÁLISE DOS RESULTADOS .....</b>                           | <b>80</b> |
| 6.1      | Implementação das funcionalidades previstas.....              | 80        |
| 6.2      | Integração com o backend.....                                 | 80        |
| 6.3      | Experiência de Utilização.....                                | 81        |
| 6.4      | Desempenho dos recursos nativos .....                         | 82        |
| 6.5      | Uniformidade das interfaces entre múltiplas plataformas ..... | 82        |
| 6.6      | Dependências Third-party.....                                 | 83        |
| 6.7      | Manutenção do código .....                                    | 84        |
| <b>7</b> | <b>CONCLUSÕES E TRABALHO FUTURO .....</b>                     | <b>85</b> |
| 7.1      | Considerações finais.....                                     | 85        |
| 7.2      | Trabalho Futuro.....  | 87        |
|          | <b>Bibliografia.....</b>                                      | <b>89</b> |
| <b>A</b> | <b>ANEXOS .....</b>   | <b>93</b> |
| a.1      | Especificação dos Casos de Uso .....                          | 93        |

---

## LISTA DE FIGURAS

---

|           |  |    |
|-----------|--|----|
| Figura 1  | Previsão do crescimento do número dispositivos (em biliões).....   | 7  |
| Figura 2  | Representação de uma arquitetura cloud para uma Smart Home.....    | 9  |
| Figura 3  | Interfaces da aplicação Apple Home.....                            | 12 |
| Figura 4  | Exemplo de um componente em React Native.....                      | 23 |
| Figura 5  | Diagrama de domínio do problema.....                               | 38 |
| Figura 6  | Casos de Uso - Gestão de Utilizadores.....                         | 39 |
| Figura 7  | Casos de Uso - Adicionar entidades.....                            | 40 |
| Figura 8  | Casos de Uso - Criar tarefas.....                                  | 41 |
| Figura 9  | Casos de Uso - Gestão das entidades.....                           | 42 |
| Figura 10 | Diagrama de Classes - Visão Geral.....                             | 44 |
| Figura 11 | Diagrama de Classes - Estatísticas e Delayed Jobs.....             | 45 |
| Figura 12 | Resposta de um pedido realizado ao openWeatherMap.....             | 49 |
| Figura 13 | Exemplo de um gráfico de consumo de uma lâmpada durante o dia..... | 60 |
| Figura 14 | Testes Unitários executados com sucesso.....                       | 66 |
| Figura 15 | Cobertura dos testes unitários.....                                | 67 |
| Figura 16 | Registo do Utilizador.....   | 71 |
| Figura 17 | Login do Utilizador.....   | 71 |
| Figura 18 | Homepage.....  | 72 |
| Figura 19 | Instalar uma casa.....   | 72 |
| Figura 20 | Visão geral das casas do utilizador.....                           | 73 |
| Figura 21 | Divisões de uma casa.....  | 73 |
| Figura 22 | Adição de novos dispositivos.....                                  | 74 |
| Figura 23 | Consola do middleware após mudar o estado de um dispositivo.....   | 74 |
| Figura 24 | Receção de pedidos no túnel do middleware.....                     | 75 |
| Figura 25 | Visão geral dos dispositivos.....                                  | 75 |
| Figura 26 | Criação de um cenário.....   | 76 |
| Figura 27 | Adição de dispositivos aos cenários.....                           | 76 |
| Figura 28 | Criação de uma Tarefa Temporal.....                                | 77 |

|           |  |    |
|-----------|--|----|
| Figura 29 | Notificação de uma tarefa cronometrada .....         | 77 |
| Figura 30 | Consola do middleware após tarefa ser executada..... | 78 |
| Figura 31 | Criação de uma Tarefa Acionada por Dispositivos..... | 78 |
| Figura 32 | Estatísticas do Utilizador .....                     | 79 |
| Figura 33 | Definições e Preferências.....                       | 79 |
| Figura 34 | User Interfaces nas diferentes plataformas.....      | 83 |

---

## LISTA DE TABELAS

---

|           |  |    |
|-----------|--|----|
| Tabela 1  | Tipos de Itens disponíveis .....                                       | 17 |
| Tabela 2  | Comparação dos critérios nas diferentes tipologias de aplicações.....  | 22 |
| Tabela 3  | Técnica de priorização MoSCoW .....                                    | 29 |
| Tabela 4  | Exemplo do formato padrão de requisitos funcionais. ....               | 30 |
| Tabela 5  | Requisitos funcionais do produto. ....                                 | 32 |
| Tabela 6  | Exemplo do formato padrão de requisitos não funcionais. ....           | 32 |
| Tabela 7  | Requisitos Não-Funcionais de Aparência .....                           | 33 |
| Tabela 8  | Requisitos Não-Funcionais de Usabilidade .....                         | 34 |
| Tabela 9  | Requisitos Não-Funcionais de Desempenho .....                          | 34 |
| Tabela 10 | Requisitos Não-Funcionais Operacionais .....                           | 35 |
| Tabela 11 | Requisitos Não-Funcionais de Manutenção e Suporte .....                | 35 |
| Tabela 12 | Requisitos Não-Funcionais de Segurança .....                           | 36 |
| Tabela 13 | Requisitos Não-Funcionais Culturais e Políticos .....                  | 36 |
| Tabela 14 | Requisitos Não-Funcionais Legais .....                                 | 36 |
| Tabela 15 | Especificação do Caso de Uso: Registo de um Utilizador .....           | 93 |
| Tabela 16 | Especificação do Caso de Uso: Autenticação de um utilizador .....      | 94 |
| Tabela 17 | Especificação do Caso de Uso: Adição de uma casa.....                  | 94 |
| Tabela 18 | Especificação do Caso de Uso: Adição de uma divisão .....              | 95 |
| Tabela 19 | Especificação do Caso de Uso: Adição de um dispositivo.....            | 96 |
| Tabela 20 | Especificação do Caso de Uso: Adição de um cenário de utilização ..... | 96 |
| Tabela 21 | Especificação do Caso de Uso: Adição de dispositivos a um cenário..... | 97 |
| Tabela 22 | Especificação do Caso de Uso: Adição de uma tarefa cronometrada.....   | 97 |
| Tabela 23 | Especificação do Caso de Uso: Adição de uma tarefa acionada.....       | 98 |
| Tabela 24 | Especificação do Caso de Uso: Consulta da meteorologia .....           | 99 |
| Tabela 25 | Especificação do Caso de Uso: Consulta de estatísticas .....           | 99 |

---

## LISTA DE LISTAGENS

---

|      |   |    |
|------|---|----|
| 2.1  | Representação de um cenário no EclipseSmartHome . . . . . | 15 |
| 4.1  | Hub - Representação de um model . . . . .                 | 50 |
| 4.2  | Hub - Representação de um controller . . . . .            | 51 |
| 4.3  | Hub - Criação de uma estatística . . . . .                | 51 |
| 4.4  | Representação do Wrapper . . . . .                        | 52 |
| 4.5  | Exemplo de utilização do Wrapper . . . . .                | 53 |
| 4.6  | Utilização da API da Câmara . . . . .                     | 55 |
| 4.7  | Utilização da API da Geolocalização . . . . .             | 56 |
| 4.8  | Utilização da API da Vibração . . . . .                   | 57 |
| 4.9  | Exemplo da utilização de Angular . . . . .                | 57 |
| 4.10 | Exemplo da aplicação do padrão Observable . . . . .       | 59 |
| 4.11 | Armazenamento local dos dados de Login . . . . .          | 60 |
| 4.12 | Tradução de texto . . . . .                               | 61 |
| 4.13 | Registo dos dispositivos dos utilizadores . . . . .       | 62 |
| 4.14 | Invocação da API para envio de notificação . . . . .      | 63 |
| 4.15 | Obtenção de API key através do middleware . . . . .       | 64 |
| 4.16 | Exemplo de um teste unitário . . . . .                    | 65 |

---

## LISTA DE ACRÓNIMOS E SIGLAS

---

**API** Application Programming Interface. 14

**DOM** Document Object Model. 23

**DSL** Domain Specific Language. 14

**HLI** Human Language Interpreter. 13

**HTTP** Hypertext Transfer Protocol. 14

**IoT** Internet of Things. 1

**MVC** Model-View-Controle. 25

**OSGi** Open Services Gateway initiative. 13

**PWA** Progressive Web App. 19, 55

**SDK** Software Development Kit. 18

**SEE** Server Sent Events. 14

**SH** Smart Home. 9

**SO** Sistema Operativo. 2

**UML** Unified Modeling Language. 37



---

## INTRODUÇÃO

---

A presente dissertação consiste no estudo das "*Aplicações híbridas para Smart Homes*" no âmbito do Mestrado em Engenharia Informática, da Universidade do Minho.

O objetivo desta dissertação consiste em estudar o comportamento das aplicações híbridas numa vertente de desenvolvimento de aplicações para Smart Homes. Esta área de aplicação da Internet of Things [Zanella et al. \(2014\)](#) começa a ser cada vez mais uma realidade nas nossas casas, contudo a informação é ainda limitada no que toca a obter a melhor solução no momento em que se pretende desenvolver uma aplicação móvel para permitir ao utilizador a gestão da sua casa.

O foco passa por estudar extensivamente as funcionalidades que devem constar num contexto de Smart Homes [Georgiev and Schlögl \(2018b\)](#), aplicá-las numa aplicação móvel, analisar o seu comportamento e por fim concluir a viabilidade da utilização desta tipologia.

Neste capítulo será efetuada uma contextualização ao tema da dissertação, a motivação que levou à escolha do mesmo e objetivos que se pretendem alcançar neste projeto.

### 1.1 ENQUADRAMENTO

Vivemos atualmente uma era de avanços tecnológicos visíveis em diferentes setores, que juntamente com a abundância e facilidade de obtenção de informação, o aumento da esperança média de vida e com a procura em reduzir esforços e recursos, nos levam à utilização de produtos que cada vez mais se tornam imprescindíveis no nosso quotidiano e onde se destacam os dispositivos IoT.

A Internet of Things representa o conceito de uma rede interligada por objetos (eletrodomésticos, roupas, edifícios, meios de transporte, entre outras) que possui um ou mais sensores capazes de recolher informação, analisá-la e transmiti-la a outros dispositivos. Este conceito tem vindo a ganhar progressivamente mais espaço no mundo tecnológico, sendo uma das suas principais utilizações a automação de tarefas em habitações – Smart Homes [Georgiev and Schlögl \(2018b\)](#).

A automação de tarefas em casas inteligentes visa melhorar a segurança, o conforto e a redução de consumos energéticos. Desta forma, a par dos dispositivos IoT, é necessário um sistema aplicacional que permita a gestão dos mesmos e que satisfaça os requisitos básicos para um utilizador de uma Smart Home, como por exemplo

definir regras e cenários de aplicação para os diferentes equipamentos da habitação. As soluções aplicacionais para interação com o utilizador podem apresentar características e funcionalidades diferentes entre si, até porque o seu desenvolvimento pode ser realizado através da utilização de diferentes tipologias: aplicações nativas, híbridas ou web. A decisão da tipologia a utilizar no desenvolvimento da aplicação é de elevada importância para o projeto e devem ser considerados diversos fatores, uma vez que um mau planeamento (que leve à mudança de paradigma durante o desenvolvimento do projeto, por exemplo) poderá resultar num aumento de custos e possíveis perdas de informação crítica.

Esta dissertação surge na evidência da necessidade das empresas em decidir um paradigma tecnológico para as suas soluções correndo o menor risco, onde serão abordadas as aplicações híbridas como uma opção preferencial a adotar. Posteriormente será desenvolvida uma aplicação híbrida, utilizando a framework “Ionic Framework” [Dunka et al. \(2017\)](#), como demonstração de uma solução para uma smart home. Esta aplicação irá consumir os serviços desenvolvidos no âmbito de uma outra dissertação de Mestrado em Engenharia Informática [de Sousa \(2017\)](#) que consistiu na construção de um middleware “cloud-based” que unifica as APIs de diferentes serviços.

## 1.2 MOTIVAÇÃO

Atualmente empresas como Apple e a Samsung desenvolvem, para o ramo das Smart Homes, aplicações apenas para iOS e Android respetivamente. Deste modo, um utilizador que deseje equipar a sua casa com equipamentos da Apple, deverá garantir que os dispositivos de todos os habitantes deverão ter este SO instalado, caso desejem fazer uso da sua SmartHome, o que limita em parte a cómoda utilização de sistemas deste tipo. Por outro lado, outras organizações que procuram desenvolver aplicações para múltiplos sistemas operativos têm o entrave de cada SO ter a sua própria linguagem de programação, ou seja requerem diferentes e diversificados programadores. Surge aqui então a necessidade do desenvolvimento de aplicações multiplataforma, ou seja, aplicações que possam ser executadas em múltiplos sistemas operativos, como é o caso das aplicações híbridas.

De modo a retirar conclusões sobre a efetividade da tipologia nesta área, pretende-se estudar as aplicações híbridas, desenvolvendo uma aplicação com as diversas funcionalidades esperadas para este tipo de produto, e posteriormente analisar o seu comportamento como possível solução para o desenvolvimento de aplicações Smart home.

## 1.3 OBJETIVOS

Como referido anteriormente, o objetivo principal desta dissertação passa por estudar as aplicações híbridas e explorar as diferentes características das mesmas, analisando essencialmente os seus benefícios na sua aplicação numa vertente de casas inteligentes.

Pretende-se que após o estudo efetuado, todo o conhecimento obtido seja aplicado através do desenvolvimento de uma aplicação móvel com a complexidade que numa aplicação em contexto empresarial é exigida, de forma a avaliar se a utilização de aplicações híbridas são ou não viáveis neste ramo. A lista de requisitos da aplicação a desenvolver seguirá um esquema de priorização, e como tal, existem funcionalidades que obrigatoriamente deverão estar presentes na solução:

- Registrar utilizadores e permitir que os mesmos configurem as suas casas, divisões e dispositivos;
- Possibilidade de alterar o estado dos sensores dos diversos equipamentos da casa;
- Criação de cenários de aplicação simultaneamente em múltiplos dispositivos;
- Criação de tarefas que são acionadas temporalmente ou por alteração do estado de um dispositivo;
- Envio de notificações para o utilizador quando as tarefas são acionadas;
- Histórico de utilização de dispositivos e gestão de custos.

Como já mencionado, será utilizado o middleware desenvolvido numa outra dissertação, no entanto espera-se que para que alguns dos requisitos acima descritos possam ser desenvolvidos, terá de ser adicionada alguma lógica no middleware implementado. Essas alterações, embora que não estejam diretamente ligadas ao tema desta dissertação, serão igualmente referidas neste documento.

#### 1.4 ESTRUTURA DO DOCUMENTO

No Capítulo 2 será documentado o estado da arte relativo aos conceitos e tecnologias que serão estudadas e analisadas no âmbito desta dissertação. Os conceitos a assimilar serão a Internet of Things, Smart Homes e tecnologias já existentes nesta área. Seguidamente será também efetuado um estudo das diferentes tipologias de aplicações existentes, dando destaque à análise das aplicações híbridas e das suas ferramentas.

No Capítulo 3 procurar-se-á relatar o processo de análise dos requisitos relativos à aplicação a desenvolver. Neste capítulo serão ainda expostos algumas diagramas obtidos de modo a descrever a arquitetura que se pretende na aplicação híbrida.

O Capítulo 4 irá servir para representar a etapa mais extensa deste projeto: a fase da implementação, seguindo com a elaboração de testes unitários e finalizando com o deploy do produto num ambiente produtivo e pronto a executar por qualquer utilizador.

No Capítulo 5 será efetuado uma pequena demonstração da aplicação, explorando algumas das funcionalidades desenvolvidas, com o intuito de exibir a aplicação híbrida concebida neste projeto.

O Capítulo 6 caracteriza-se pela fase onde a aplicação desenvolvida é analisada e são retirados os devidos resultados, face ao que é proposto demonstrar: a eficácia de uma aplicação híbrida como uma solução para uma casa inteligente.

Por fim, no Capítulo 7 serão apresentadas as conclusões do trabalho desenvolvido, onde serão efetuadas as considerações finais da dissertação e serão apresentadas propostas para trabalho futuro.

---

## ESTADO DA ARTE

---

O Estado da Arte tem como objetivo apresentar alguns conceitos inerentes ao tema abordado nesta dissertação. Neste capítulo irão constar todas as temáticas inerentes ao desenvolvimento de uma aplicação híbrida nesta área das tecnologias de informação. Serão numa primeira fase abordados os tópicos relativos à Internet of Things e Smart Homes, sendo que para estas últimas serão apresentadas algumas aplicações no mercado existentes nesta área. Posteriormente serão estudadas as diferentes tipologias de aplicações existentes de forma a realizar uma comparação entre as mesmas.

### 2.1 INTERNET OF THINGS

#### 2.1.1 Definição

A Internet das Coisas (do inglês *Internet of Things*) emergiu com o avanço em algumas áreas tecnológicas como microeletrónica e comunicação e é um tópico que se está a tornar cada vez mais explorado na área das tecnologias da informação. É um conceito que tem o potencial de mudar não só a forma como vivemos o nosso dia-a-dia, mas também a forma como se desenvolvem outros setores, como por exemplo a indústria automóvel, construção civil ou nas energias [Osseiran et al. \(2017\)](#).

Não existe uma definição concreta para descrever o conceito de Internet of Things, mas de forma geral, pode-se definir este conceito como uma rede de objetos, quaisquer que eles sejam, com a capacidade computacional e de comunicação de se ligarem à internet, de forma a recolher dados e a transmiti-los entre si, funcionando como fornecedores de diferentes serviços [Zanella et al. \(2014\)](#).

A expressão "Internet of Things" é formada por dois termos chave – "**Internet**" e "**Things**" [Madakam et al. \(2015\)](#). O primeiro termo define o meio sob o qual este conceito está desenvolvido, a Internet. A Internet pode-se caracterizar como um sistema global de redes conectadas entre si, com o objetivo de servir biliões de utilizadores pelo mundo inteiro. Na IoT, a Internet é considerada o fator chave por possibilitar que os objetos troquem entre si os dados recolhidos pelos mesmos, através de diversos protocolos. Estes objetos referidos levam ao segundo termo da expressão, as *Things* (Coisas). As *Things*, de acordo com a *International Telecommunication Union* (ITU), são

“objetos do mundo físico ou do mundo virtual da informação , que são capazes de serem identificados e integrados em redes de comunicação U (2012).

### 2.1.2 *Objetivos da IoT*

A evolução constante de tecnologias como dispositivos eletrônicos, comunicações, sensores, smartphones, sistemas incorporados, cloud computing, virtualização da rede, entre outros, permitem que constantemente existam mudanças e melhorias nesta área Patel et al. (2016).

Pode-se considerar que o principal objetivo na utilização de dispositivos de Internet of Things é melhorar a qualidade de vida de quem na sociedade usufrui das suas tecnologias. Tendo isto em conta, os objetos físicos que são utilizados por todos nós no dia a dia, desde máquinas de café, televisões, lâmpadas e automóveis, estão a tender para uma evolução em que a relação Homem-Máquina será cada vez menos frequente para dar lugar à relação Máquina-Máquina Matthews (2019). Nesta relação, as máquinas comunicam entre si para satisfazer as necessidades do ser humano, sem a interferência deste. Para que esta relação possa acontecer, é então necessário proporcionar alguma inteligência aos objetos e meios para que eles compartilhem entre si as informações que recolhem, sendo que graças aos avanços tecnológicos hoje em dia é possível implementar estas condições em qualquer tipo de objetos, desde os mais pequenos e simples, aos maiores e complexos.

Outro objetivo primordial da IoT é também alargar a diversidade de objetos inteligentes e que possam comunicar entre si, tanto dentro da mesma rede, como ao longo de toda a Internet. Desta forma é possível aumentar a possibilidade de criar sinergias em cenários de utilização, desde ambientes industriais, áreas da saúde, domótica, entre outros. Esta comunicação entre diferentes objetos é assim um dos principais desafios da IoT devido à diversidade de dispositivos, plataformas (HomeKit, AllJoyn, etc) e de protocolos de comunicação (WiFi, Bluetooth).

### 2.1.3 *Crescimento da IoT*

É certo que o conceito de Internet of Things surgiu já há algum tempo e que o seu uso em áreas como a indústria e domótica é já uma realidade, contudo espera-se que a sua evolução seja constante, expandindo a sua utilização a passos largos para outras áreas, mas também aprimorando as áreas em que já se recorre à utilização de dispositivos IoT. Para que os diversos dispositivos de IoT tenham a possibilidade de comunicar entre si de forma mais controlada, eficaz e inteligente, novas tecnologias estão também a emergir, como por exemplo a *Cloud Computing*, *Big Data* e a *Computação Distribuída* Sayed Ali Ahmed and Kamal (2017).

Atualmente é difícil encontrar estudos que indiquem que o crescimento da IoT irá estagnar em breve, sendo que as previsões Statista (2018) indicam que este crescimento se irá traduzir em números como cerca de 30 mil milhões de dispositivos conectados em 2020, e 75 mil milhões em 2025 (Figura 1), o que irá levar a uma maior motivação de empresas, investigadores e cidadãos em apostar neste mercado Chin et al. (2019).

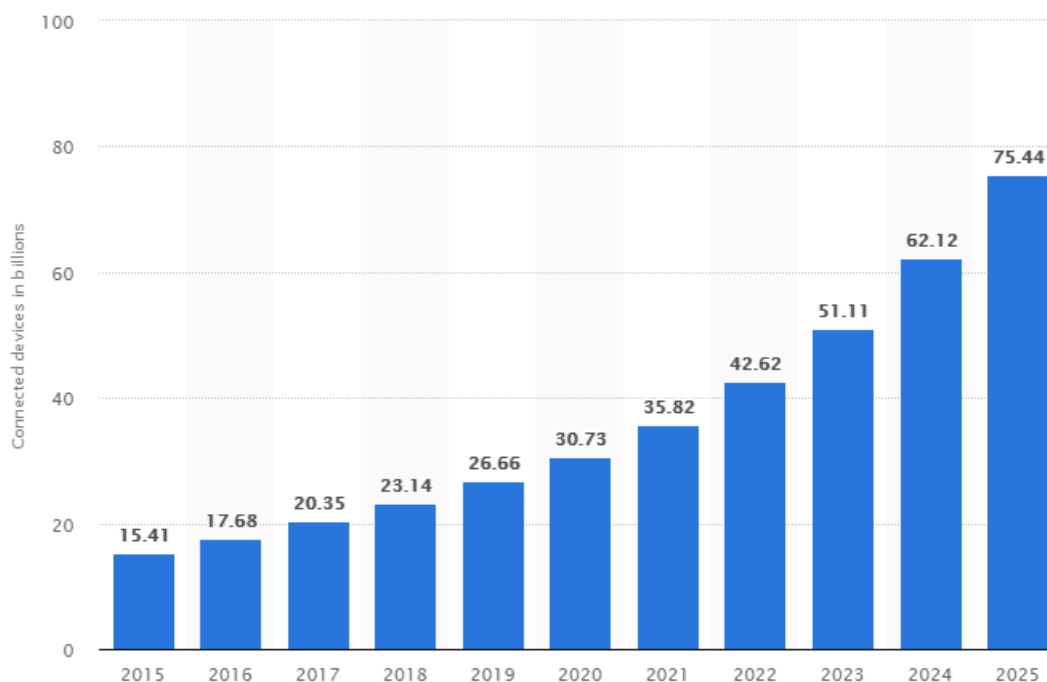


Figura 1: Previsão do crescimento do número dispositivos (em bilhões)

O aumento dos dispositivos de IoT conectados entre si promove o aparecimento de novas aplicações para que se possa monitorizar o seu comportamento na rede, como por exemplo sistemas de monitorização de máquinas industriais ou de monitorização de florestas, o que levará cada vez mais ao aparecimento de novos desafios no mundo tecnológico.

## 2.2 SMART HOME

Smart Home é apenas uma das diversas áreas de aplicação da Internet of Things. Este termo designa uma residência que está equipada com dispositivos com tecnologia de ligação à rede (por exemplo Wi-Fi, Bluetooth ou Zigbee), que pode ser remotamente ser acedida, controlada e monitorizada [Georgiev and Schlögl \(2018b\)](#). O seu principal objetivo é responder às necessidades dos seus habitantes, possibilitando aos seus utilizadores um conforto, segurança e entretenimento através da gestão dos equipamentos IoT. Este conceito abre assim novos horizontes para que os dispositivos que usamos diariamente na nossa rotina, se tornem em equipamentos eletrónicos e inteligentes. Estes equipamentos podem ser sistemas de controlo de climatização, controlo de iluminação, câmaras de segurança, TVs, frigoríficos, entre muitos outros [Domb \(2019\)](#).

O conceito de ter uma casa a executar automaticamente as nossas tarefas diárias tem ganho cada vez mais popularidade, sendo que hoje em dia já se torna um hábito ter pelo menos um dispositivo inteligente nas habitações. É relativamente simples adicionar dispositivos inteligentes nas nossas casas que permitam que a nossa vida seja mais facilitada, levando também a uma redução de custos, energia e tempo.

### 2.2.1 *Arquitetura de Smart Homes*

Com a evolução das Smart Home diversas arquiteturas foram aparecendo de forma a satisfazer as necessidades do utilizador e a tornar o produto mais eficiente. Tendo em consideração que um dos objetivos desta dissertação é realizar uma aplicação que possibilite o controlo remoto de uma Smart Home independentemente da sua localização, entre as diversas arquiteturas existentes, será dado o destaque a uma que satisfaça esta necessidade.

Segundo Katuk [Katuk et al. \(2018\)](#) uma possível arquitetura que permita a utilização remota de uma Smart Home (Figura 2) é composta pelos seguintes elementos:

#### *Dispositivos IoT*

Como já foi referido na secção anterior, os dispositivos IoT são o componente fundamental numa Smart Home. São equipamentos domésticos com tecnologias que permitem a comunicação à rede, dando a possibilidade ao utilizador de mudar o estado dos seus sensores através do uso de uma aplicação. Esta alteração de estado dos sensores, por norma, pode ser realizada no instante ou no futuro, através de tarefas de automação.

Contudo, cada um destes dispositivos não consegue comunicar entre si sem que exista algo que os conecte. Para que os diferentes dispositivos partilhem informação mutuamente é necessário que estes estejam conectados não só à internet, mas também entre si através de um gateway.

#### *Gateway*

Um gateway de uma Smart Home – também designado de Hub - é um dispositivo cujo objetivo é conectar os diferentes dispositivos na rede e controlar a comunicação entre eles [Doan et al. \(2018\)](#). Funciona como uma ponte que traduz os diferentes protocolos de comunicação dos equipamentos, isto porque nem sempre os equipamentos são das mesmas marcas, ou usam os mesmo protocolos de comunicação (WI-FI, Bluetooth, etc.), e como tal é necessário a existência desta interface para compatibilizar a sua comunicação.

#### *Aplicação*

O Hub e os dispositivos são monitorizados através de uma aplicação (móvel ou num terminal instalado nas infra-estruturas) que permite a gestão de cada um dos dispositivos conectados na rede. Estas aplicações proporcionam ao utilizador a possibilidade de monitorizar qualquer dispositivo na rede com o mínimo esforço possível, através



de um simples clique e sem ter de se deslocar ao dispositivo para alterar o seu estado. Adicionalmente estas aplicações permitem a criação de cenários de utilização e a automação de tarefas, como por exemplo, no momento em que um utilizador entra em casa, as luzes são ligadas e a temperatura é ajustada aos valores definidos pelo mesmo.

### Protocolos

Já vimos que os dispositivos comunicam entre si através de um Hub, contudo é graças aos diferentes protocolos de comunicação que isto é possível. Os protocolos definem a forma como os estados dos dispositivos são enviados de equipamento para equipamento com o objetivo de executar uma ação.

Os diversos dispositivos presentes na rede nem sempre são concebidos pelas mesmas empresas e como tal os protocolos que utilizam não são necessariamente os mesmos. Existem atualmente diferentes protocolos de comunicação, desde Wi-Fi, Bluetooth, Zigbee ou Z-Wave [Sarawi et al. \(2017\)](#) e a tendência é este número aumentar face à evolução tecnológica e à diversidade dos equipamentos [Nascimento \(2016\)](#). É importante ter este ponto em consideração no momento da construção de uma solução para SH, preparando toda a arquitetura para receber futuramente novos tipos de equipamentos.

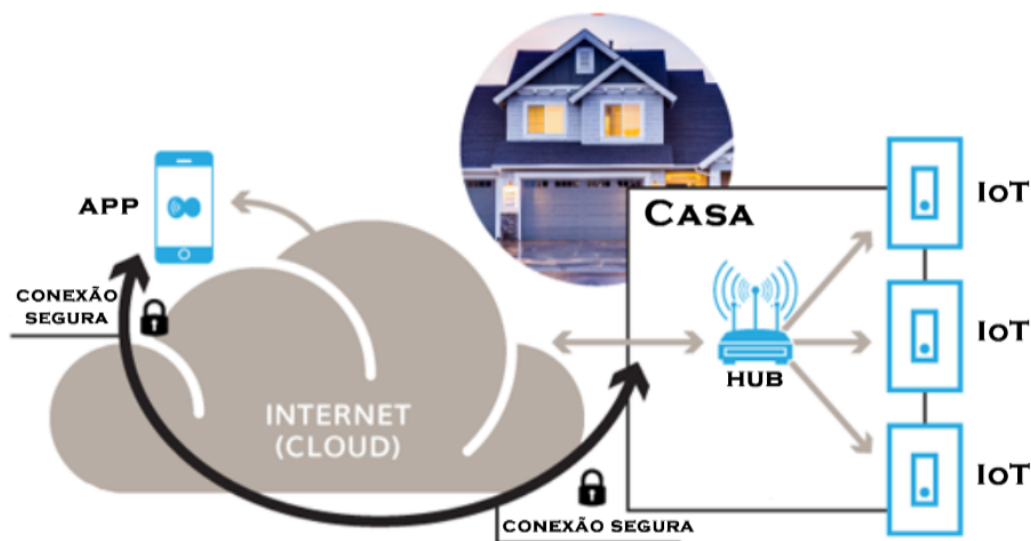


Figura 2: Representação de uma arquitetura cloud para uma Smart Home

### 2.2.2 Utilizadores

As soluções para Smart Homes são pensadas e desenvolvidas tendo em conta dois fatores muito importantes para os utilizadores: o primeiro é otimizar o conforto, bem-estar e qualidade de vida dos habitantes e o segundo é proporcionar melhores condições de acessibilidade tanto para idosos como para pessoas com incapacidades físicas [Katuk et al. \(2018\)](#). Apesar disso, o utilizador olha para este conceito de casas inteligentes e automatizadas ainda com alguma desconfiança, sentindo-se ameaçado com possíveis ataques maliciosos [Georgiev and Schlögl \(2018a\)](#).

Do ponto de vista económico, apesar da utilização de Smart Homes permitir aos utilizadores uma redução de custos a longo prazo, o facto dos equipamentos IoT terem ainda um preço consideravelmente elevado, é ainda um entrave para a maioria dos utilizadores [Georgiev and Schlögl \(2018a\)](#). Estes consideram que a relação redução de custos/investimento não é ainda compensada ao ponto de tornarem a sua casa numa casa inteligente.

Quanto à aplicação para a gestão das Smart Homes, por ser o meio pelo qual os utilizadores podem realmente interagir com os dispositivos da rede, deverá ser de fácil e intuitiva utilização, tendo também em conta questões de acessibilidade para diferentes utilizadores [Hargreaves and Wilson \(2013\)](#).

É assim necessário estudar aquilo que os utilizadores esperam encontrar nas aplicações no momento em que desejam automatizar as suas casas. Num estudo realizado pela [Alarm.com Alarm.com \(2016\)](#) com 1022 proprietários de habitações, foram recolhidos os seguintes dados:

- 88% dos inquiridos desejam a existência de uma única aplicação que controle todos os dispositivos que possuam.
- 91% dos inquiridos esperam que esses mesmos dispositivos possam automaticamente trabalhar em conjunto.
- 88% desejam ser notificados de alterações de estado dos seus dispositivos.
- 93% consideram importante economizar energia, principalmente em termostatos.

Deste modo, considera-se fundamental pensar numa solução que satisfaça as necessidades gerais dos inquiridos, que irá refletir as necessidades básicas que utilizadores de Smart Homes esperam que sejam proporcionadas.

### 2.2.3 Tecnologias

Neste tópico serão estudadas tecnologias já existentes no mercado e que servem como solução para Smart Homes. É um tópico fundamental para a realização desta dissertação, pois irão ser revistas as funcionalidades das tecnologias e as respectivas arquiteturas, características que serão fundamentais mais à frente, no desenvolvimento da solução.

Inicialmente será explorada uma tecnologia comercial e seguidamente uma *open-source*, existentes no mercado e que correspondem a duas das aplicações mais populares por quem quer monitorizar uma Smart Home, segundo um questionário realizado IEEE (2016).

### *Apple HomeKit*

O Apple HomeKit é uma framework que tem o objetivo de conectar e permitir a comunicação de equipamentos destinados a casas inteligentes e que suportem o protocolo de acessórios da HomeKit Apple (2019b). Permite aos utilizadores encontrar e configurar esses mesmos acessórios, e criar um conjunto de regras para os controlar, como por exemplo, configurar a aplicação de forma a aumentar a temperatura da casa ou acender a iluminação de algumas zonas da mesma quando os sensores de movimento detetam a chegada do utilizador. Estes dispositivos são armazenados nos dispositivos iOS do utilizador que são sincronizados através do iCloud para outros dispositivos do mesmo sistema operativo. Outra característica do Apple HomeKit é a integração com a *Siri*, o assistente pessoal incluído nos dispositivos móveis da Apple, que permite o controlo da aplicação por voz, sem a necessidade do contacto do utilizador diretamente na aplicação móvel.

O Homekit da Apple para além da framework que permite a programadores construir aplicações para iOS, possui também uma aplicação móvel associada designada de Home. Este kit surge no meio de outros "kit's" que a Apple desenvolveu e fornece aos utilizadores, sendo que para além deste existe também o ResearchKit e CareKit (ambos relacionados com a área da saúde), GamePlayKit (relacionado com o desenvolvimento de jogos), entre outros.

Os serviços que esta framework fornece permitem ao utilizador adicionar ao seu setup qualquer acessório que seja compatível com o HomeKit, isto é, apenas aplicações revistas pela Apple e presentes no App Store (plataforma de distribuição digital da Apple) podem ser usadas neste sistema. Atualmente, segundo a Apple são cerca de 100 marcas que já produzem acessórios compatíveis com o HomeKit, contudo um número em constante crescimento. Estes acessórios são revistos e aprovados pela Apple com o intuito de garantir a melhor experiência de utilização ao cliente e sobretudo, segurança.

Uma característica do HomeKit é a automatização do processo de descoberta de dispositivos, isto é, acessórios que já tenham sido configurados num dispositivo iOS, ficam guardados na base de dados, e são automaticamente reconhecidos num outro dispositivo móvel, como um iPad. Outro benefício deste kit é a garantia que utilizador não tem de se preocupar sobre a forma como a comunicação entre o HomeKit e os equipamentos é realizada. Como já referido anteriormente, o HomeKit possui também a característica de permitir a integração da *Siri* para a realização de tarefas através de comandos áudio.

Na Figura 3 podemos ver um exemplo de algumas interfaces da aplicação Apple Home:



Figura 3: Interfaces da aplicação Apple Home

A Apple fornece através da sua documentação oficial [Apple \(2019a\)](#) a informação necessária para que os programadores possam utilizar esta framework para implementar as suas próprias soluções. Através da consulta dessa mesma documentação, é possível resumir a terminologia utilizada da seguinte forma:

- **Homes** – pode ser considerada como o todo, isto é, representam uma casa do utilizador, mas também podem representar anexos no exterior por exemplo. Um utilizador pode possuir mais que uma “home” na aplicação, e monitorizá-las de forma individual e independente umas das outras.
- **Rooms** - representam divisões dentro da casa (*home*), como por exemplo salas, cozinhas ou quartos. As *rooms*, necessitam de ter nomes diferentes umas das outras para que seja possível atribuir regras ou ordens através da Siri, como por exemplo “ligar as luzes da Sala e desligar as luzes da Cozinha”.
- **Zones** – para além de poder definir regras por *homes* ou por *rooms*, é possível definir por zonas da casa, como por exemplo andares, ou seja, definir regras para o rés do chão e para o primeiro andar, de modo isolado e independente. Após definidas na aplicação, o utilizador pode utilizar comandos como, “Aumentar a temperatura do 1º andar para 25°C”, que irá definir a temperatura do andar para o valor definido, independentemente da temperatura das outras zonas.
- **Accessories** - são os dispositivos físicos (equipamentos IoT) instalados nas casas e que devem ser conectados ao HomeKit como por exemplo uma lâmpada, uma câmara ou uma TV. Os acessórios comunicam com a Cloud de forma a receber instruções, responder a instruções e fornecer o estado dos seus serviços.

- **Services** – correspondem aos serviços oferecidos por um dispositivo (podem ser vistos como funcionalidades), como por exemplo, um serviço para fechar o portão de uma garagem. Cada dispositivo pode oferecer vários serviços para serem controlados pelo utilizador.
- **Characteristics** – é um atributo de um serviço que terá o seu estado correspondente. Os seus estados podem corresponder a simples on/off ou a níveis, como por exemplo "Controlar o grau de luminosidade de uma lâmpada ligada" ou "Definir a temperatura do ar condicionado".
- **Actions** – uma ação corresponde ao ato de alterar os estados de um dispositivo através da utilização de um serviço, como por exemplo, "Ajustar a velocidade de um ventilador".
- **Scenes** – é um conjunto de ações com o objetivo de espoletar um ou mais serviços (alterando o estado) dos acessórios, de forma a satisfazer possíveis cenários de utilização que o utilizador deseje. Por exemplo, é possível criar um cenário "Good Night" que desliga todas as lâmpadas de casa e fecha os estores das janelas.
- **Automation** – é a forma como se comportam os equipamentos quando expostos a certas situações, como quando o sensor de movimento deteta movimento ou em diferentes horas do dia (dia/noite) . Regras para a automatização podem ser definidas, como por exemplo "Acender a iluminação do jardim quando o sol se põe".

### *Eclipse SmartHome*

O Eclipse SmartHome é, tal como o Apple HomeKit, uma framework direcionada para o desenvolvimento de aplicações em torno das casas inteligentes. Contudo, ao contrário do Apple Homekit, é uma framework *open-source*, o que permite a qualquer programador modificar e partilhar o código contribuindo para a evolução constante da framework e consequentemente do conceito de *Smart Home*. Como qualquer framework, o Eclipse SmartHome fornece aos programadores e empresas recursos, que neste caso se referem a *user interfaces* pré definidas, regras de automação, mecanismos de persistência e interoperabilidade de dispositivos [SmartHome \(2019a\)](#).

Através da utilização desta framework é possível desenvolver projetos relacionados com casas inteligentes, possibilitando aos programadores concentrarem-se unicamente em ambientes heterogêneos, isto é, soluções que lidam com a integração de diferentes protocolos ou padrões, ao contrário do Apple HomeKit. O objetivo é fornecer um acesso uniforme a dispositivos e facilitar as interações entre eles. Esta estrutura consiste num conjunto de pacotes *OSGi* que permitem um desenvolvimento em múltiplas plataformas.

Esta framework caracteriza-se por apresentar as seguintes características:

- **Audio & Voice** – Através do [HLL](#) existe a possibilidade de controlar a aplicação através de comandos de voz, uma característica bastante útil para os utilizadores que usufruam de uma *Smart Home*, por permitir uma interação natural e facilitada com o sistema.

- **Rest API** – O Eclipse SmartHome fornece uma Rest *API*, que pode ser usada para se comunicar com outro sistema, pois permite o acesso de leitura a itens e estados dos mesmo, atualizações de status ou envio de comandos para itens, usando para o efeito mensagens *HTTP* para comunicar. Possui também suporte para eventos enviados pelo servidor (*SEE*) para que as aplicações desenvolvidas recebam notificações das alterações de estado dentro de um sistema.
- **Textual Configuration** – Permite a configuração dos elementos de um sistema totalmente à base de texto, usando para o efeito *DSL's* para os diferentes tipos de elementos usados. De seguida é apresentado um exemplo para a configuração de uma *Thing*:

```
yahooweather : weather : berlin[location = 638242]
```

- **Internationalization** – Todos os textos desta framework podem ser internacionalizados usando arquivos com propriedades *Java 118N*. É utilizado para cada idioma um arquivo específico com notação de *postfix*, que consiste num código de idioma. Os códigos para especificação do idioma em alemão e em francês são os seguintes:

```
yahooweather_de.properties
```

```
yahooweather_fr.properties
```

- **Rules** – As regras são um elemento essencial neste conceito de *Smart Home*, e através desta framework a sua utilização é uma possibilidade. Estas regras são úteis para controlar dispositivos com base no estado de outros presentes na mesma rede, como por exemplo fechar as janelas de casa a partir de uma hora pré-definida. A sua sintaxe é definida da seguinte forma:

```
ON item_id statechanged
IF item_id.state == desired_value
THEN item_id2.state = desired_value2
```

Através da sintaxe podemos observar que um item identificado como *item\_id2* altera o seu estado com o *desired\_value2* quando o estado do *item\_id* é igual ao *desired\_value*.

As rules são divididas em três componentes chave:

- Triggers - Evento que espoleta o acionamento de uma regra;
  - Conditions - Definem a condição que é verificada pelo trigger.
  - Actions - Especifica as ações que devem ser espoletadas caso as condições sejam verdadeiras.
- **Scenes** – Como referido anteriormente, um cenário é o conjunto de estados que um ou mais dispositivos deve alterar em simultâneo após sua ativação numa Smart Home. Um exemplo de um cenário pode ser um "Movie Scene", que liga automaticamente a televisão, desliga todas as luzes e fecha os estores das janelas. No Eclipse SmartHome, o conceito é recriado de forma diferente do Apple Homekit: um cenário é definido dentro da secção de uma rule (Listagem 2.1).

```

"uid": "light.scene1",
"name": "IndoorLightScene",
"tags": [
  "night",
  "scene"
],
"description": "A night scene for indoor lights.",
"triggers": [],
"conditions": [],
"actions": [
  {
    "id": "ItemPostCommandActionID1",
    "type": "core.ItemCommandAction",
    "configuration": {
      "itemName": "corridorLightItem",
      "command": "ON"
    }
  },
  {
    "id": "ItemPostCommandActionID2",
    "type": "core.ItemCommandAction",
    "configuration": {
      "itemName": "roomLampItem",
      "command": "ON"
    }
  }
]

```

Listagem 2.1: Representação de um cenário no EclipseSmartHome

Neste exemplo (2.1) é representado um cenário para ligar a iluminação, mais propriamente as lâmpadas do quarto e do corredor.

- **Graphic Interfaces** - Esta framework oferece também interfaces pré definidas, cada uma com características e modos de implementação diferentes, mas todas possuem elementos para o desenvolvimento de interfaces gráficos, desde recursos CSS e JavaScript, ícones ou widgets.

O Eclipse SmartHome divide o seu sistema em dois modelos, o modelo físico, e o modelo funcional. Enquanto o modelo físico se preocupa com os equipamentos envolventes no conceito da *Smart Home*, o modelo funcional preocupa-se com a forma como as coisas acontecem, isto é, com a lógica do sistema.

A framework, tal como o Apple Homekit, fornece através da sua documentação oficial [SmartHome \(2019b\)](#) toda a informação para que os programadores e empresas que desejem utilizar esta framework open-source desenvolvam os seus sistemas. Deste modo, pode-se resumir a terminologia desta framework da seguinte forma:

- **Things** - Entidades que podem ser adicionadas fisicamente a um sistema com o objetivo de fornecer funcionalidades ao sistema. É importante destacar que as things não precisam de ser necessariamente equipamentos, mas também podem representar um *Web Service*, por exemplo. Fornecem as suas funcionalidades através de um conjunto de *Channels*, que serão falados mais à frente. Visto que são parte do modelo físico, são relevantes para o processo de configuração e configuração do sistema, mas não para a parte lógica do mesmo.
- **Channels** - Podem ser considerados como as funcionalidades que cada *Thing* pode oferecer, como por exemplo, uma *Thing* que seja uma lâmpada, tem um channel para a luminosidade e outro para a cor que ela assume.
- **Items** - Os itens representam a funcionalidade que é utilizada pela aplicação (principalmente em interfaces gráficas ou na automatização). Os itens têm um estado e são usados através de eventos.

Os seguintes tipos de itens estão atualmente disponíveis (ordem alfabética):

| Nome do Item    | Descrição                       | Tipos de Comandos                     |
|-----------------|---------------------------------|---------------------------------------|
| <b>Color</b>    | Informação da cor (RGB)         | OnOff, IncreaseDecrease, Percent, Hsb |
| <b>Contact</b>  | Estado de Armazenamento do Item | OpenClose                             |
| <b>DateTime</b> | Armazena data e hora            | -                                     |
| <b>Dimmer</b>   | Item com um valor percentual    | OnOff, IncreaseDecrease, Percent      |
| <b>Group</b>    | Agrupar itens em grupos         | -                                     |



|                      |   |  |
|----------------------|---|--|
| <b>Image</b>         | Possui os dados de uma imagem                   | -  |
| <b>Location</b>      | Armazena as coordenadas GPS                     | Point                                      |
| <b>Number</b>        | Armazena os valores                             | Decimal                                    |
| <b>Player</b>        | Permite controlar players, por exemplo de áudio | PlayPause, NextPrevious, RewindFastforward |
| <b>Rollershutter</b> | Usado normalmente em persianas                  | UpDown, StopMove, Percent                  |
| <b>String</b>        | Armazena texto                                  | String                                     |
| <b>Switch</b>        | Usado normalmente para lâmpadas                 | OnOff                                      |

Tabela 1: Tipos de Itens disponíveis

- **Categories** - São usadas para fornecer informações de meta-dados sobre *Channels* de *Things*. Existem *Categories* como Car, Door, HVAC ou Speaker.
- **Inbox & Discovery** – Permite que sejam descobertos dispositivos automaticamente na rede ou através de uma API. No Eclipse SmartHome são enviados os resultados da descoberta para a inbox da aplicação, de forma a que posteriormente sejam configurados e utilizados.

## 2.3 APLICAÇÕES MÓVEIS

As aplicações móveis representam um dos elementos mais populares das Tecnologias de Informação e Comunicação e definem-se como software que visa satisfazer necessidades e realizar tarefas dos utilizadores [Islam and Mazumder \(2010\)](#). Estas tarefas podem corresponder a funcionalidades como o envio de mensagens, visualização de vídeos ou edição de imagens.

O primeiro smartphone a aparecer no mercado foi desenvolvido em 1993 pela *International Business Machines Corporation* (IBM) e com ele chegaram também as primeiras aplicações. Eram 10 aplicações incorporadas no smartphone, e passados mais de 25 anos esses números são bem diferentes: segundo as informações para as principais *Stores*, existem à data no *Google Play* 2.570.000 aplicações, na *Apple App Store* 1.840.000, 669.000 aplicações na *Windows Store*, e por fim, 489.000 na *Amazon AppStore* [Statista \(2020\)](#). As previsões são para que estes números continuem a aumentar ao longo dos próximos anos, face à aposta cada vez mais recorrente das empresas no desenvolvimento de aplicações móveis para os seus produtos e serviços.

As aplicações móveis cada vez mais são parte do nosso cotidiano, permitindo que todos nós possamos realizar múltiplas tarefas através da utilização dos nossos dispositivos. Contudo, este aumento de popularidade, a par dos avanços tecnológicos, apesar de permitir e incitar o desenvolvimento de soluções inovadoras, leva à existência de uma divergência das plataformas. Existem atualmente vários sistemas operativos e estes divergem não só nas linguagens de programação em que estão construídos, mas também nas ferramentas que são desenvolvidas para estes SO's. Esta característica pode resultar num problema no momento em que as empresas pretendem desenvolver aplicações para diferentes plataformas.

Portanto, no processo de desenvolvimento de aplicações móveis, a escolha do tipo de aplicação a desenvolver é um ponto extremamente importante, sendo necessário considerar fatores como o objetivo do produto e o seu público-alvo e questões tecnológicas como os sistemas operativos e funcionalidades a serem abordadas, de forma a atribuir um peso para estas diferentes variáveis e a perceber qual será a melhor solução. Neste ponto, destacam-se como principais tipologias as aplicações Web, as aplicações nativas e por fim, as aplicações híbridas. Nesta secção será dado o destaque às aplicações híbridas, estudando de que forma é possível potenciar as suas características em soluções de smart homes.

### 2.3.1 Aplicações nativas

O conceito de aplicações nativas designa o desenvolvimento de aplicações com o objetivo de que estas sejam executadas em dispositivos ou plataformas específicas (iOS, Android, Windows, etc), sendo apenas comercializadas pelas *Stores* das respetivas empresas. As aplicações nativas são mais populares em smartphones, mas o seu desenvolvimento acontece também para desktops, smartTV's e gadget's. Este tipo de aplicações é construído através de linguagens de programação específicas, como o Swift para iOS, Java para Android ou C# para Windows Phone Blanco (2016).

As aplicações nativas garantem ao utilizador um melhor desempenho, uma maior rapidez na sua utilização por estas terem acesso direto às APIs nativas dos dispositivos. A experiência do utilizador é também melhorada graças ao desenvolvimento através de SDK nativos, o que proporciona o desenvolvimento de User Interfaces consistentes e agradáveis. Estes dois pontos são mais visíveis em aplicações mais pesadas, como por exemplo jogos com gráficos intensos. Adicionalmente, outro grande benefício na utilização deste tipo de aplicações é a capacidade de utilização das mesmas em modo offline (na maioria dos casos), ao contrário das aplicações Web que requerem em todo o momento conexão à Internet para sua utilização.

Contudo, é para as empresas que surge o maior entrave face a esta tipologia. Como já foi referido, a linguagem de programação destas aplicações irá depender da plataforma para que estas serão desenvolvidas. Caso o modelo de negócio das empresas passe por lançar o mesmo produto para diferentes sistemas operativos, levará à necessidade de desenvolver aplicações distintas, necessitando de uma quantidade maior de recursos.

### 2.3.2 Aplicações Web

As aplicações Web, ao contrário das nativas, não requerem download e instalação e a sua utilização é realizada via browser, como por exemplo o Chrome ou Mozilla Firefox. Estas aplicações necessitam de ligação à Internet durante todo o tempo de utilização, e são desenvolvidas em tecnologias como JavaScript, CSS e HTML5, ou seja, as linguagens padrão da Web. Contrariamente às aplicações nativas, as aplicações Web não possuem acesso aos recursos de hardware do dispositivo, como por exemplo a câmara e GPS.

Estas aplicações podem então ser designadas como websites com um layout e funcionalidades desenvolvidas a pensar na utilização móvel, dando assim ao utilizador a sensação de utilização de uma aplicação nativa Blanco (2016). Uma das metodologias de desenvolvimento de aplicações Web é a utilização de PWA's, conceito introduzido pela Google em 2015 Tandel and Jamadar (2018). Seguindo as diretrizes desta metodologia, uma aplicação Web utiliza as tecnologias Web mais recentes com o objetivo de diminuir as diferenças encontradas entre com os outros tipos de aplicações, adicionando desta forma funcionalidades de aplicações nativas e híbridas, como por exemplo aceder a algumas funcionalidades em modo offline e a capacidade de utilização de notificações push.

Esta tipologia de aplicação é vista como uma solução para as empresas quando os recursos que estas possuem são mais limitados, dado que as estas aplicações possuem maior facilidade de desenvolvimento, o que leva a um tempo e custos de desenvolvimento inferiores, comparativamente às outras tipologias.

### 2.3.3 Aplicações Híbridas

Híbrido, por definição é algo derivado de fontes heterogéneas ou composta de elementos de tipos diferentes. As aplicações híbridas aparecem aqui como um meio termo entre as duas tipologias anteriormente descritas, apresentando um comportamento nativo, mas desenvolvido através de tecnologias Web Blanco (2016).

Estas aplicações necessitam primeiramente de serem descarregadas a partir das lojas oficiais de distribuição dos softwares (App Store, Google Play, etc.) e posteriormente instaladas. As aplicações são implementadas num container nativo que utiliza um objeto WebView móvel, exibindo desta forma o conteúdo Web graças à utilização de tecnologias como CSS, JavaScript e HTML5.

Tal como nas aplicações nativas, também é possível utilizar recursos de hardware de um dispositivo (Câmara, GPS ou acelerómetro) em virtude de ferramentas existentes que favorecem esta comunicação entre a WebView (páginas ou simples componentes web da aplicação) e a plataforma. Estas ferramentas, na maior parte delas, não representam soluções oficiais das principais plataformas (iOS e Android), e apresentam-se como soluções terceiras como é o caso do Apache Cordova.

Para as empresas que recorrem a este tipo de aplicações, a principal vantagem é possibilidade de uma só aplicação poder ser usada pelos vários sistemas operativos disponíveis, reutilizando o mesmo código para diversas plataformas, sendo assim mais eficazes em termos de custo (desde programadores a tempo de desenvolvimento).

Este desenvolvimento pode ser realizado através da utilização de diversas frameworks existentes, deste Ionic Framework<sup>1</sup>, NativeScript<sup>2</sup>, React Native<sup>3</sup> ou Xamarin<sup>4</sup>.

#### 2.3.4 Análise comparativa entre o desenvolvimento de aplicações móveis Nativas, Web e Híbridas

Segue-se uma análise comparativa entre as diferentes abordagens para o desenvolvimento de aplicações móveis tendo por base um conjunto de características convergentes com um conjunto de critérios que foram identificados durante o estudo das diferentes tipologias nos pontos anteriores, utilizando para a devida comparação os trabalhos de Nunkesser [Nunkesser \(2018\)](#), Vilček e Jakopec [Vilček and Jakopec \(2017\)](#) e Jobe [Jobe \(2013\)](#). Os critérios para a análise comparativa entre as abordagens de desenvolvimento são os seguintes:

- **Desempenho** - uma das principais vantagens da utilização de ferramentas de desenvolvimento nativo face a qualquer outra tipologia, reside no fato destas aplicações apresentarem um melhor desempenho, com uma utilização mais leve e fluída. Código nativo, como Java ou Swift, é executado mais rapidamente do que por exemplo JavaScript, o que leva a um reduzido desempenho em aplicações Web e híbridas. Esta diferença é facilmente observada em aplicações com gráficos intensos, como é o caso dos jogos.
- **Custo e tempo de desenvolvimento** – O custo de desenvolvimento de uma aplicação pode ser determinado através de dois fatores: horas requeridas para a construção da aplicação e o custo de cada uma dessas horas. O número de horas necessárias irá depender das funcionalidades que se pretende adicionar, mas também dos sistemas operativos. Enquanto que numa aplicação Web/Híbrida é necessária apenas uma aplicação, nas aplicações nativas será necessária uma por cada SO. Para além disso, os programadores especializados em desenvolvimento com tecnologias Web têm um custo menor que programadores especializados em tecnologias Android e iOS.
- **Proteção de dados** – No que toca a segurança e proteção de dados, o desenvolvimento de aplicações nativas garantem uma maior segurança da informação. A arquitetura nativa é protegida por diversas camadas no sistema operativo, o que torna difícil a sua exploração de forma maligna. Por outro lado, as aplicações híbridas, devido à dependência de tecnologias terceiras que raramente são atualizadas, possuem vulnerabilidades na segurança. A utilização de tecnologias Web abrem também espaço para ataques web conhecidos como a manipulação de dados e *SQL Injection*.
- **User Experience** - Por norma, as aplicações nativas oferecem uma interface mais responsiva e fluída comparativamente às outras tipologias, isto graças ao acesso direto a todas as APIs dos dispositivos e possibilidade de utilização destas aplicações em modo offline. Outra característica favorável neste contexto diz

1 Ionic Framework - <https://ionicframework.com>

2 NativeScript - <https://nativescript.org>

3 React Native - <https://reactnative.dev>

4 Xamarin - <https://dotnet.microsoft.com/apps/xamarin>

respeito ao facto dos utilizadores possuírem comportamentos diferentes quando apreciam uma aplicação: é mais comum ser descarregada uma aplicação e deixá-la instalada no telemóvel (no caso de aplicações nativas e híbridas) do que armazenar aplicações Web nos marcadores.

- **Dependências terceiras** - O desenvolvimento nativo, por norma, não irá necessitar de dependências externas, graças à capacidade de aceder diretamente às APIs nativas do dispositivo, como a câmara ou serviços de localização. Por outro lado, nas aplicações Híbridas, para que consigam aceder a estas funcionalidades, são utilizadas por norma ferramentas open-source como é o caso de Xamarin e Cordova. Sempre que estas dependências externas são atualizadas e fornecem novas funcionalidade, é aumentada a probabilidade da ocorrência de bugs, o que pode ser um entrave no desenvolvimento das aplicações.
- **Manutenção do código** - A não existência dependências “third-party” significa uma redução de bugs e dessa forma a uma redução de custos de manutenção. Contudo, quando estas acontecem, envolvem uma modificação no código existente para as diferentes aplicações nativas existentes para cada sistema operativo. As aplicações Híbridas e Web, por terem o código centralizado numa só aplicação apresentam à partida um menor custo de manutenção. Na perspetiva do utilizador, é também importante ter em conta que no desenvolvimento nativo e híbrido, atualizações de software irão requerer que o utilizador atualize as aplicações através das *App Stores*.
- **Funcionalidades** – Como já falado anteriormente, as aplicações nativas possuem uma maior facilidade em aceder às funcionalidades do dispositivo, contudo aplicações híbridas e até mesmo aplicações Web conseguem também este acesso (não na totalidade), embora que principalmente nestas últimas, resulte num pior desempenho. É também com as aplicações nativas que são implementadas com uma maior qualidade funcionalidades de última geração como a Realidade Virtual, Realidade Aumentada e Inteligência Artificial. Web Apps têm vindo também a ter avanços nas suas funcionalidades, como é o caso da adição de notificações “push” e utilização da “vibração” do hardware.
- **Conexão à internet** – O que acontece quando o utilizador não tem acesso à internet? Aqui as aplicações nativas levam vantagem. Na sua instalação e atualização das mesmas, as aplicações nativas irão necessitar de internet, contudo, durante a sua utilização, não necessita obrigatoriamente de acesso à internet, a não se que existam funcionalidades que o exijam. Estas aplicações conseguem facilmente guardar informação no dispositivo local sem ter de utilizar internet para o fazer. Por outro lado, as aplicações híbridas requerem internet quase na totalidade do tempo, e as web necessitam durante toda a sua utilização.
- **Distribuição via App Stores** - Neste ponto, as aplicações nativas e as aplicações híbridas são muito semelhantes, ao contrário das aplicações Web. As primeiras requerem um descarregamento através das App Stores correspondentes. Estas App Stores para além de funcionarem como um bom local para encontrar novas aplicações no mercado, garantem uma maior segurança e confiança na aplicação, pois as aplicações

são verificadas e testadas antes de entrarem no mercado, de forma a garantir o cumprimento dos termos e condições das mesmas. A desvantagem do mercado das App Stores é que estas ficam com uma comissão dos valores ganhos através dos downloads efetuados. Contudo, existe uma maior visibilidade das empresas com as aplicações presentes nas stores comparativamente às aplicações Web.

- **Suporte ao desenvolvimento** - As aplicações nativas são as pioneiras no desenvolvimento de aplicações, sendo que estas comunidades desde cedo apareceram com conteúdos diversos como vídeos e blogs com tutoriais, guias de desenvolvimento e fóruns onde programadores apresentam e/ou respondem a dúvidas. Todas as outras tipologias, por serem mais recentes, apresentam comunidades menores, embora exista também diverso suporte para tecnologias Web, como por exemplo JavaScript, como para tecnologias de desenvolvimento híbrido, como é o caso de PhoneGap.

|   | Aplicações Nativas    | Aplicações Híbridas   | Aplicações Web |
|---|-----------------------|-----------------------|----------------|
| <b>Desempenho</b>                       | Ótimo                 | Bom                   | Razoável       |
| <b>Custo e tempo de desenvolvimento</b> | Demorado              | Mediano               | Rápido         |
| <b>Protecção de dados</b>               | Bom                   | Razoável              | Razoável       |
| <b>Experiência do utilizador</b>        | Excelente             | Ótima                 | Boa            |
| <b>Dependências terceiras</b>           | Reduzido              | Elevado               | Mediano        |
| <b>Manutenção do código</b>             | Razoável              | Boa                   | Ótimo          |
| <b>Funcionalidades</b>                  | Ótimo                 | Bom                   | Razoável       |
| <b>Conexão à internet</b>               | Opcional <sup>a</sup> | Opcional <sup>a</sup> | Obrigatório    |
| <b>Distribuição via app store</b>       | Existente             | Existente             | Inexistente    |
| <b>Suporte ao desenvolvimento</b>       | Excelente             | Bom                   | Ótimo          |

Tabela 2: Comparação dos critérios nas diferentes tipologias de aplicações

<sup>a</sup> Poderão existir funcionalidades que impliquem o acesso à Internet

## 2.4 DEFINIÇÃO DAS TECNOLOGIAS HÍBRIDAS

### 2.4.1 *React Native*

React Native é uma framework *open-source* desenvolvida pelo Facebook em 2015 e é uma das tecnologias que permite o desenvolvimento híbrido de aplicações. Esta framework surge após Mark Zuckerberg, CEO da empresa, ter exprimido publicamente que um dos principais problemas da aplicação em 2010 seria o excessivo foco no consumo de HTML5, o que seria na altura um entrave para a utilização da rede social em aplicações móveis Warren (2012). As principais vantagens da utilização desta framework comparativamente a outras similares, como por exemplo Ionic Framework, passam pelo melhor desempenho e inferiores consumos de CPU e bateria Krispinsson (2017).

Através da utilização desta framework torna-se possível o desenvolvimento de apenas uma aplicação que pode ser utilizada por qualquer sistema operativo sem comprometer a experiência de utilização, fornecendo um conjunto de componentes que são diretamente mapeados nas API's nativas dos dispositivos.

O conceito chave de React Native é a utilização de componentes, que representam pequenas partes da interface, como por exemplo os campos para login de uma aplicação. O objetivo principal deste conceito é reutilizar os diversos componentes ao longo da aplicação. Esta framework não manipula diretamente o DOM: existem processos em background que interpretam os diferentes componentes diretamente nas API's nativas, possibilitando um desenvolvimento mais rápido e facilitado Gill (2018).

```
import React, { useState } from 'react';
import {
  View, Text, Button, StyleSheet
} from 'react-native';

function App() {
  const [count, setCount] = useState(0);

  return (
    <View styles={styles.container}>
      <Text>You clicked {count} times</Text>
      <Button
        onPress={() => setCount(count + 1)}
        title='Click me'
      />
    </View>
  );
}

// React Native Styles
const styles = StyleSheet.create({
  flex: 1,
  justifyContent: 'center',
  alignItems: 'center'
});
```

Figura 4: Exemplo de um componente em React Native

Na Figura 4 está representado um exemplo de um componente nesta framework onde o *output* é um contador para o número de vezes que um botão foi clicado. O componente retorna a interface a apresentar ao utilizador, sendo também adicionado o *style* definido para o componente. Essa interface é desenvolvida através da utilização de JSX, que permitem a inserção de HTML diretamente no JavaScript.

#### 2.4.2 Ionic framework

Ionic Framework apareceu em Novembro de 2013 e rapidamente se tornou uma das principais frameworks de desenvolvimento de aplicações híbridas utilizadas pelas empresas, por esta permitir a criação de funcionalidades inovadoras em aplicações móveis, devido ao facto desta estar construída em cima de duas importantes tecnologias [Bhupesh and Joshi \(2019\)](#):

- **AngularJS** – Uma das principais frameworks para JavaScript, permitindo assim aproveitar as melhores práticas de desenvolvimento desta linguagem e construir aplicações com uma arquitetura bem especificada, garantindo desta forma uma melhor integração com web services e outros componentes.
- **Apache Cordova** – é também uma framework *open-source* e funciona como uma peça do software que conecta os recursos nativos e aplicação web numa só aplicação, renderizando os componentes web numa WebView nativa.

A framework é caracterizada pela utilização de Componentes Web (HTML, CSS e JavaScript) e possui diversos componentes pré-definidos (desde botões, tabs, popups), mas também permite a integração com outras frameworks, como Angular ou React, permitindo assim que os desenvolvedores construam os componentes “à sua medida”. [Dunka et al. \(2017\)](#) A integração com tipo de tecnologias permite também aceder aos recursos nativos do dispositivo, possibilitando a criação de funcionalidades de última geração, o que proporciona ao utilizador uma experiência de utilização inovadora.

A utilização desta framework oferece ao desenvolvimento híbrido alguns benefícios, desde a fácil adoção da tecnologia graças à utilização de componentes web, a existência de inúmeros componentes pré-configurados facilmente customizáveis e também um desempenho superior àquilo que é expectável para o desenvolvimento híbrido, consequência da utilização do apache Cordova. Comparativamente à framework React Framework, o seu desempenho é ligeiramente inferior, contudo o tempo de desenvolvimento é reduzido e a estrutura/design da aplicação é mais elegante. É uma framework bastante aceite pelos programadores, pois para além das diversas vantagens já descritas, a estrutura dos projetos em Ionic está bem idealizada, tornando o desenvolvimento das aplicações mais facilitado.



### 2.4.3 Apache Cordova

Também conhecido como PhoneGap, Apache Cordova é uma das ferramentas imprescindíveis aquando do desenvolvimento de aplicações híbridas. No momento da sua criação, em 2009, a empresa canadense Nitobi designou esta tecnologia como PhoneGap, sendo que esta foi desenvolvida com o intuito de possibilitar que às aplicações web a utilização das API's nativas. Em 2011 a empresa foi adquirida pela Adobe Systems, que posteriormente lançou uma versão open source da tecnologia, passando a tecnologia a designar-se Apache Cordova.

Esta framework fornece as ferramentas necessárias aos programadores para que estes sem capazes de desenvolver aplicações utilizando componentes Web. Isto acontece graças a existência de WebView's, que possibilita que os componentes (como por exemplo botões ou tabs) sejam apresentados dentro de um container nativo, podendo assim executar código JavaScript ou executar páginas HTML.

Para além desta framework possibilitar a execução de aplicações Web dentro de uma aplicação nativa, o Apache Cordova também oferece API's JavaScript que permitem aceder a uma variedade de funcionalidades nativas dos dispositivos.

Apesar desta framework garantir estas ferramentas, a mesma necessita de trabalhar em conjunto com outras tecnologias para que as aplicações a desenvolver possam apresentar uma User Interface apelativa e com um aspeto nativo. Desta forma, a integração com Ionic Framework é uma solução bastante plausível, podendo recorrer aos mais de 100 componentes pré-definidos que a framework possui, desde animações, botões, listas, tabs e scrolling, reduzindo assim o tempo e custo de desenvolvimento.

### 2.4.4 Angular

Angular é uma framework *open-source* que favorece o desenvolvimento de aplicações usando componentes Web, construída pelas equipas da Google em Setembro de 2016. Foi inicialmente lançada como "Angular 2", mas posteriormente renomeada para apenas "Angular" face à confusão que gerava nos desenvolvedores.

Angular aparece como uma rescrição de raiz da sua versão anterior, AngularJS, depois de terem sido identificadas as necessidades dos desenvolvedores ao longo do tempo. AngularJS seguia um padrão arquitetural designado de MVC, contrastando com Angular que aparece com uma arquitetura construída à volta do conceito de componentes Kasagoni (2017). Um componente pode ser designado como um bloco de desenvolvimento que pode ser reutilizado ao longo da aplicação. Normalmente os componentes são divididos em 3 membros fundamentais: o template HTML que diz respeito à "forma" que o componente vai ter na página, o CSS onde pode ser definido o estilo dos diferentes elementos da página, e por fim, uma classe JavaScript que possui a lógica comportamental que o componente pode assumir. Outros elementos importantes desta framework para além dos componentes, são os templates, diretivas, módulos, injeção de dependências e ferramentas de infraestrutura Kasagoni (2017).

Esta framework foi desenvolvida em TypeScript, também esta uma linguagem de programação *open-source*, desenvolvida pela Microsoft. A utilização de TypeScript é quase obrigatória quando se trata de aplicações JavaScript mais complexas, por possuir características como a possibilidade de utilização de classes e interfaces. A utilização de TypeScript permite também que os IDE's (ambientes de desenvolvimento integrado) indiquem erros comuns durante a escrita de código, facilita os desenvolvedores e reduz o tempo de desenvolvimento.

É uma das principais frameworks oficiais de integração para servir de frontend à framework Ionic Framework, a par de React e Vue.js Saks (2019). A sua comunidade online é bastante alargada, desde blogs, vídeos e tutoriais e outros tipos de fóruns, o que facilita a sua utilização por programadores com menor experiência.

## 2.5 ARQUITETURA DO MIDDLEWARE DESENVOLVIDO

Como já referido anteriormente, esta dissertação dá seguimento ao trabalho desenvolvido numa dissertação anterior, onde foi desenvolvido um middleware que permite a comunicação entre os diferentes componentes presentes numa casa inteligente. Este middleware, devido à sua complexidade, possui uma arquitetura composta por vários elementos concisos e completos, cada um deles com os respetivos objetivos e funcionalidades.

Uma das peças do sistema é o Hub, ou gateway, que corresponde a um hardware com a capacidade de descobrir vários tipos de dispositivos do utilizador e comunicar com os mesmos. O Hub irá constantemente contactar com a API através do protocolo HTTP, até porque o mesmo é *stateless*, isto é, não armazena qualquer tipo de informação, sendo que esta capacidade estará apenas ao nível da API. Neste projeto académico, este Hub é simulado através da utilização de um Raspberry Pi.

Outro elemento importante desta arquitetura é a API, que funciona como o motor de regras do middleware. Desenvolvido na linguagem de programação *Ruby*, através da framework *Ruby on Rails*, este componente contém toda a estrutura comportamental inerente às entidades do sistema e também é quem comunica com a base de dados para armazenar a informação. A lógica da maior parte das entidades, como utilizadores, divisões e dispositivos é composta essencialmente por operações CRUD, sendo que é adicionada uma maior complexidade na lógica das tarefas automatizadas.

Esta arquitetura é também composta por um *wrapper* que converte as invocações realizadas à API utilizando HTTP, para métodos ou funções na linguagem da aplicação que irá consumir estes serviços, que corresponde neste caso, a JavaScript. Esta técnica permite a reutilização de funções, o que diminui a complexidade da aplicação cliente. No *wrapper* é utilizada uma biblioteca designada *axios*<sup>5</sup>, que corresponde a um cliente HTTP que permite a interação com os endpoints da API. O *wrapper* pode assim oferecer uma classe que contém vários módulos, um para cada endpoint da API.

Por fim, é necessário que existam dispositivos IoT que sejam descobertos pelo Hub. Neste ponto, para simular o funcionamento destes dispositivos no mundo real, são utilizados vários tipos de simuladores. Grande parte destes

---

5 Axios - <https://github.com/axios/axios>

simuladores fornecem uma API HTTP Restful, que caracteriza um padrão habitual na indústria das Smart Homes, contudo são também utilizados outros tipos de protocolos.

---

## ANÁLISE E CONCEÇÃO

---

Após uma leitura aprofundada sobre conceitos inerentes a este projeto e tecnologias que serão utilizadas no desenvolvimento de uma aplicação híbrida para Smart Home, é importante analisar no que deve consistir a solução a desenvolver, descrevendo uma lista de requisitos que a aplicação deve apresentar e concebendo modelos da interface, tendo em consideração os requisitos propostos. Todo este processo pode ser dividido em duas fases:

- **Análise de requisitos:** é aqui que será analisada toda a informação recolhida de forma a recolher os pontos principais, e essencialmente, definir uma lista de requisitos funcionais e não funcionais para a solução.
- **Conceção:** na fase de conceção serão desenhados diferentes diagramas e construídos modelos que irão representar a proposta de solução a implementar.

Esta fase de desenvolvimento representa o momento em que os desenvolvedores e o cliente mais comunicam entre si de forma a que possam debater a proposta de solução e definir em conjunto o desenho da solução final. Tendo em conta que esta dissertação não se enquadra numa situação em que desenvolve uma solução para um cliente em específico, ir-se-á recorrer a estudos e tecnologias semelhantes no mercado que revelam o que é mais procurado pelos utilizadores numa aplicação nesta temática, de modo a garantir as funcionalidades básicas numa aplicação neste contexto.

### 3.1 ANÁLISE DE REQUISITOS

O planeamento de qualquer projeto é uma fase importante para que se possa entender o problema, com o principal objetivo de definir uma solução correta, evitando tomadas de decisão que não levem ao sucesso do projeto. Assim, será imperativo definir um minucioso planeamento da aplicação a desenvolver no âmbito desta dissertação,

A análise de requisitos é fundamental para que o desenvolvimento e implementação do projeto ocorra da melhor forma possível. Assim, este capítulo irá servir para descrever não só os requisitos funcionais que se referem às funcionalidades e comportamentos que a aplicação deverá apresentar mas também os requisitos não funcionais referentes às propriedades ou características da mesma.

Este processo de levantamento de requisitos será efetuado tendo em consideração a técnica de priorização designada *MoSCoW*, uma das principais utilizadas para o efeito. [Hudaib et al. \(2018\)](#). A designação desta técnica diz respeito a um acrónimo utilizado para representar grupos de prioridade, permitindo desta forma atribuir graus de importância aos requisitos apresentados [Ahmad et al. \(2017\)](#), como representado na Tabela 3.

| Grau de priorização | Representação | Descrição  |
|---------------------|---------------|--|
| <b>Must</b>         | <b>M</b>      | Requisitos críticos que obrigatoriamente devem estar presentes               |
| <b>Should</b>       | <b>S</b>      | Requisitos importantes e que devem ser considerados                          |
| <b>Could</b>        | <b>C</b>      | Requisitos desejáveis mas não necessários                                    |
| <b>Won't</b>        | <b>W</b>      | Requisitos que não serão desenvolvidos, mas serão considerados para o futuro |

Tabela 3: Técnica de priorização MoSCoW

O levantamento dos requisitos do sistema foi efetuado através da pesquisa previamente realizada sobre conceitos relativos a este projeto e a tecnologias similares, que demonstram o que os utilizadores consideram indispensável numa aplicação para a gestão de Smart Homes. É importante considerar que existem características e funcionalidades que estas aplicações devem apresentar, por forma a satisfazer os principais objetivos das mesmas e proporcionar aos utilizadores uma utilização satisfatória. No entanto, no âmbito desta dissertação nem todas irão ser implementadas e serão deixadas para trabalho futuro, mas para que a aplicação seja semelhante às atualmente comercializadas será definida uma lista de requisitos funcionais e não funcionais o mais completa possível.

### 3.1.1 *Requisitos funcionais*

Os requisitos funcionais descrevem as funcionalidades que a aplicação deverá proporcionar aos utilizadores finais, procurando caracterizar a forma como o sistema se comporta. Estes requisitos podem ser vistos como afirmações/declarações destas mesmas características, sem qualquer ambiguidade de interpretação, e nunca mencionando aspetos técnicos ou de interface.

O formato mais consensual para descrever estes requisitos divide-se em 3 partes: (1) – o sujeito, representando a quem está associado o requisito, (2) – o verbo, ou a ação que é esperada que aconteça através do requisito e (3) – a descrição ou o corpo do requisito. Na Tabela 4 é representado um exemplo de construção de um requisito, seguindo este padrão.

|                  |                                |
|------------------|--------------------------------|
| <b>Sujeito</b>   | O utilizador...                |
| <b>Verbo</b>     | ...adiciona...                 |
| <b>Descrição</b> | ...comentários às fotografias. |

Tabela 4: Exemplo do formato padrão de requisitos funcionais.

De forma a serem descritas todas as características e interações entre os diferentes sistemas e o ambiente que o software a desenvolver deve implementar, foi elaborada uma lista com todos os requisitos funcionais que devem ser desenvolvidos:

| Nº | Requisito   | Prioridade |
|----|---|------------|
| 1  | O sistema deverá permitir que os utilizadores se registem                       | M          |
| 2  | O sistema deverá permitir que os utilizadores façam login                       | M          |
| 3  | O sistema deverá permitir que o utilizador registe as suas casas                | M          |
| 4  | O sistema deverá permitir que o utilizador registe as divisões das suas casas   | M          |
| 5  | O sistema deverá permitir que o utilizador registe os seus dispositivos         | M          |
| 6  | O sistema deverá permitir a criação de cenários de utilização                   | M          |
| 7  | O sistema deverá poder adicionar dispositivos e respetivos estados aos cenários | M          |
| 8  | O sistema deverá permitir a criação de tarefas temporais                        | M          |
| 9  | O sistema deverá permitir a criação de tarefas acionadas                        | M          |
| 10 | O sistema deverá permitir listar todas as casas do utilizador                   | M          |
| 11 | O sistema deverá fornecer uma visão geral para cada uma das casas do utilizador | S          |
| 12 | O sistema deverá listar as divisões de cada casa                                | M          |
| 13 | O sistema deverá listar todos os dispositivos de cada casa                      | M          |
| 14 | O sistema deverá listar todos os cenários de cada casa                          | M          |
| 15 | O sistema deverá listar todas as tarefas de cada casa                           | M          |

|    |  |   |
|----|--|---|
| 16 | O sistema deverá permitir alterar o estado dos diferentes dispositivos                 | M |
| 17 | O sistema deverá permitir aplicar cenários   | M |
| 18 | O sistema deverá permitir editar a informação do utilizador                            | S |
| 19 | O sistema deverá permitir editar informação sobre cada casa                            | S |
| 20 | O sistema deverá permitir carregar uma fotografia de perfil para a casa                | C |
| 21 | O sistema deverá permitir editar informação sobre as divisões                          | S |
| 22 | O sistema deverá permitir editar a informação sobre os cenários                        | S |
| 23 | O sistema deverá permitir editar a informação sobre as tarefas                         | S |
| 24 | O sistema deverá permitir eliminar casas   | S |
| 25 | O sistema deverá permitir eliminar divisões  | S |
| 26 | O sistema deverá permitir eliminar cenários  | S |
| 27 | O sistema deverá permitir remover dispositivos dos cenários                            | S |
| 28 | O sistema deverá permitir eliminar tarefas   | S |
| 29 | O sistema deverá permitir eliminar a sua conta   | W |
| 30 | O sistema deverá apresentar a meteorologia para a localização do utilizador            | S |
| 31 | O sistema deverá apresentar estatísticas de utilização para os diferentes dispositivos | S |
| 32 | O sistema deverá apresentar estatísticas sobre as casas e divisões das mesmas          | S |
| 33 | O sistema deverá apresentar estatísticas de utilização de tarefas e cenários           | S |

|    |   |   |
|----|---|---|
| 34 | O sistema deverá permitir o envio de notificações para o utilizador | C |
| 35 | O sistema deverá permitir a execução via widgets                    | W |

Tabela 5: Requisitos funcionais do produto.

Numa primeira fase de desenvolvimento, pelo facto da lista de requisitos ser demasiado extensa, serão desenvolvidos apenas os requisitos definidos com "M", requisitos que foram considerados indispensáveis para qualquer aplicação Smart Home e para que esta dissertação pudesse apresentar conclusões o mais reais e fidedignas possíveis. O processo de desenvolvimento seguirá com a implementação dos requisitos marcados como "S" e posteriormente os requisitos "C". Por ser uma área de aplicação onde se pode divergir com bastante facilidade nas funcionalidades a implementar na aplicação, foram adicionados também requisitos "W", com o objetivo de serem concretizados em projetos futuros.

### 3.1.2 Requisitos não funcionais

Os requisitos não funcionais correspondem a um conjunto de propriedades e restrições impostas para o desenvolvimento do sistema com o intuito de destacar aspetos mais técnicos da aplicação, como por exemplo a sua rapidez, as cores utilizadas, a linguagem de programação, entre outros. Enquanto um requisito funcional descreve o que o sistema fará, o requisito não funcional define como o fará.

Existem atualmente diferentes sistemas de categorização destes requisitos. Nesta dissertação será a usada a proposta de Robertson [Robertson and Robertson \(2013\)](#) - uma classificação dividida em oito tipos de requisitos não funcionais: (1) aparência, (2) usabilidade, (3) desempenho, (4) operacionais, (5) manutenção e suporte, (6) segurança, (7) culturais e políticos e (8) legais.

Existe também para os requisitos não funcionais uma proposta padrão para a sua escrita, dividida em 2 membros: (1) - o sujeito, que representa sobre quem está associado o requisito e (2) - a qualidade ou propriedade a ser integrada no sistema. A Tabela 5 corresponde a um exemplo da construção de um requisito, seguindo este padrão.

|                              |   |
|------------------------------|---|
| <b>Sujeito/Entidade</b>      | A linguagem de programação do sistema ... |
| <b>Qualidade/Propriedade</b> | ...deverá ser Java 8.                     |

Tabela 6: Exemplo do formato padrão de requisitos não funcionais.

Da mesma forma que foram listados os requisitos funcionais, serão também listados os requisitos não funcionais, divididos pelas categorias acima mencionadas:



*Requisitos de Aparência*

Estes requisitos são referentes ao aspeto visual dos diferentes componentes da aplicação.

| Nº | Requisito  | Prioridade |
|----|--|------------|
| 1  | O sistema deve ser atrativo para todos os utilizadores   | M          |
| 2  | Todos os ícones a utilizar deverão ser coloridos e agradáveis  | S          |
| 3  | As cores dos estados dos dispositivos deverão ser intuitivas   | M          |
| 4  | Os elementos seleccionados, como por exemplo ícones e botões, deverão ser visivelmente diferenciados | M          |
| 5  | Todos os gráficos a apresentar deverão ser simples e de fácil análise                                | M          |
| 6  | O tipo de letra a usar será Helvetica Neue   | C          |
| 7  | O sistema deverá permitir a que o utilizador selecione um tema preferencial                          | C          |

Tabela 7: Requisitos Não-Funcionais de Aparência

*Requisitos de Usabilidade*

Os requisitos de usabilidade descrevem as propriedades a existir no sistema de forma a garantir uma utilização agradável.

| Nº | Requisito  | Prioridade |
|----|--|------------|
| 8  | A navegação entre os diferentes ecrãs deverá ser fácil e simples   | S          |
| 9  | A navegação dentro dos menus será efetuada através de "swipes"   | S          |
| 10 | As interfaces deverão ser responsivas e devidamente preparadas para serem apresentadas em qualquer dispositivo | M          |
| 11 | As funcionalidades da aplicação deverão ser intuitivas e de fácil aprendizagem                                 | S          |
| 12 | Deverão existir atalhos para as funcionalidades "core" do projeto  | C          |

|    |   |   |
|----|---|---|
| 13 | Deverá existir <i>Loading Screens</i> para que o utilizador entenda que a informação se encontra a carregar | S |
| 14 | A autenticação dos utilizadores ficarão guardadas em cache durante algumas horas                            | S |
| 15 | Deverão existir mensagens de erro   | S |

Tabela 8: Requisitos Não-Funcionais de Usabilidade

*Requisitos de Desempenho*

Estes requisitos definem aspetos relacionados com o desempenho da aplicação, isto é, com a rapidez de execução das funcionalidades, capacidade de armazenamento, entre outros.

| Nº | Requisito   | Prioridade |
|----|---|------------|
| 16 | A apresentação dos dados das pesquisas não deverão demorar mais do que 5 segundos para 90% das solicitações       | S          |
| 17 | A resposta do sistema deverá ser rápida o suficiente para evitar interrupção no fluxo de pensamento do utilizador | S          |
| 18 | A aplicação deverá estar disponível 24 horas por dia  | S          |
| 19 | As funcionalidades da aplicação deverão estar disponíveis independentemente da localização do utilizador          | M          |
| 20 | A aplicação deverá permitir uma fácil instalação independentemente da experiência dos utilizadores                | C          |
| 21 | A informação sobre a meteorologia deverá ser em real-time   | S          |

Tabela 9: Requisitos Não-Funcionais de Desempenho

*Requisitos Operacionais*

Os requisitos operacionais definem aspetos relativos com o sistema em que a aplicação está embebida.

| Nº | Requisito | Prioridade |
|----|-----------|------------|
|----|-----------|------------|

|    |   |   |
|----|---|---|
| 22 | A aplicação deverá funcionar nas versões mais recentes de qualquer sistema operativo  | S |
| 23 | O sistema irá necessitar de acesso à internet para a maioria das suas funcionalidades | S |
| 24 | As notificações serão recebidas após conexão à internet                               | C |

Tabela 10: Requisitos Não-Funcionais Operacionais

*Requisitos de Manutenção e suporte*

Estes requisitos descrevem as características da aplicação para que possa ser mantida no futuro.

| Nº | Requisito  | Prioridade |
|----|--|------------|
| 25 | O código fonte deverá estar comentado para facilitar a sua compreensão   | C          |
| 26 | A aplicação deverá ser submetida a manutenção sempre que necessário  | S          |
| 27 | Em caso de manutenção deverá ser garantido que as atualizações são realizadas em horários que causem o menos impacto ao utilizador | M          |
| 28 | A aplicação deverá estar preparado para adaptar a necessidades do negócio  | S          |

Tabela 11: Requisitos Não-Funcionais de Manutenção e Suporte

*Requisitos de Segurança*

Os requisitos de segurança descrevem a forma como é garantido o acesso aos utilizadores e são evitadas falhas de segurança.

| Nº | Requisito  | Prioridade |
|----|--|------------|
| 29 | A aplicação deverá garantir que apenas utilizadores registados e autenticados tenham acesso às funcionalidades | M          |

|           |   |          |
|-----------|---|----------|
| <b>30</b> | O sistema deverá garantir a proteção de dados do utilizador                   | <b>M</b> |
| <b>31</b> | O sistema deverá garantir que em caso de falha mantém a integridade dos dados | <b>M</b> |

Tabela 12: Requisitos Não-Funcionais de Segurança

*Requisitos culturais e políticos*

Nestes requisitos são contempladas as características relacionadas com aspetos culturais e políticos.

| <b>Nº</b> | <b>Requisito</b>  | <b>Prioridade</b> |
|-----------|---|-------------------|
| <b>32</b> | O sistema deverá permitir ao utilizar configurar o seu idioma preferencial.           | <b>W</b>          |
| <b>33</b> | A aplicação não deverá conter textos ou ícones que possam ofender qualquer utilizador | <b>M</b>          |
| <b>34</b> | As unidades de medida devem seguir o Sistema Internacional de Unidades                | <b>M</b>          |

Tabela 13: Requisitos Não-Funcionais Culturais e Políticos

*Requisitos Legais*

Estes requisitos referem-se a fatores relacionados com leis, regras e normas que devem ser introduzidas na aplicação.

| <b>Nº</b> | <b>Requisito</b>  | <b>Prioridade</b> |
|-----------|---|-------------------|
| <b>35</b> | Em toda a fase de desenvolvimento e manutenção da aplicação deverão ser utilizadas ferramentas open-source ou para estudantes | <b>M</b>          |

Tabela 14: Requisitos Não-Funcionais Legais

## 3.2 CONCEÇÃO

Após serem levantados os requisitos funcionais e não funcionais, segue-se a fase de conceção, que é nada mais do que a modelação desses requisitos em diferentes diagramas. Estes diagramas possibilitam aos arquitetos um melhor entendimento do fluxo do sistema e do comportamento das funcionalidades a desenvolver, de forma a cumprir os requisitos definidos. É uma fase imprescindível para os arquitetos e programadores, para que estes percebam como conceber e programar a arquitetura do produto com uma maior rapidez e qualidade.

Tendo em conta que este projeto é um seguimento de uma outra dissertação, é necessário avaliar através dos requisitos o que deve ser acrescentado no backend e respetivos serviços que esta aplicação irá consumir. Deste modo, foram elaborados diferentes diagramas onde será destacado o estado da arquitetura atual e o que se pretende adicionar.

### 3.2.1 Modelo de Domínio

O modelo de domínio é uma representação visual dos componentes do produto que incluem as suas entidades e as suas relações, fornecendo deste modo uma visão geral das classes conceituais do problema, e é através dele que se começa a desenvolver a arquitetura da aplicação. Este modelo permite assim descrever de uma forma mais sintetizada o domínio analisado, descrevendo os principais processos e a respetiva lógica para implementar a solução.

Para a construção destes diagramas UML são analisados os requisitos previamente definidos e identificados possíveis entidades nos mesmos. Sendo que estas classes definem conceitos essenciais ao problema e não especificações da solução, os modelos de domínio à partida não são alterados em aplicações distintas que apresentam soluções para o mesmo problema.

Depois de identificar as entidades e como estas se relacionam entre si é necessário indicar a multiplicidade destes relacionamentos, que podem variar nas seguintes formas:

- A multiplicidade ocorre num número de ocorrências exatas (p.e. 1);
- A multiplicidade ocorre num intervalo exato de ocorrências (por exemplo 0..1);
- A multiplicidade ocorre num intervalo não determinado de ocorrências, utilizando um \* para representar a indeterminação (por exemplo 0..\*).

Para este projeto foi extremamente importante a elaboração de um modelo de domínio para que fosse possível identificar as entidades a acrescentar ao trabalho previamente desenvolvido na dissertação anterior. Foram adicionadas na arquitetura quatro novas entidades: as Definições e Estatísticas do utilizador, os dados meteorológicos referentes à localização do utilizador e as Divisões. Com esta última entidade, os dispositivos deixaram de ter uma

relação direta com as Casas, para passarem a pertencer às Divisões, sendo que estes se relacionam com a respetiva Casa. Estas e as restantes classes e as respetivas relações podem ser visualizados no diagrama de domínio (Figura 5).

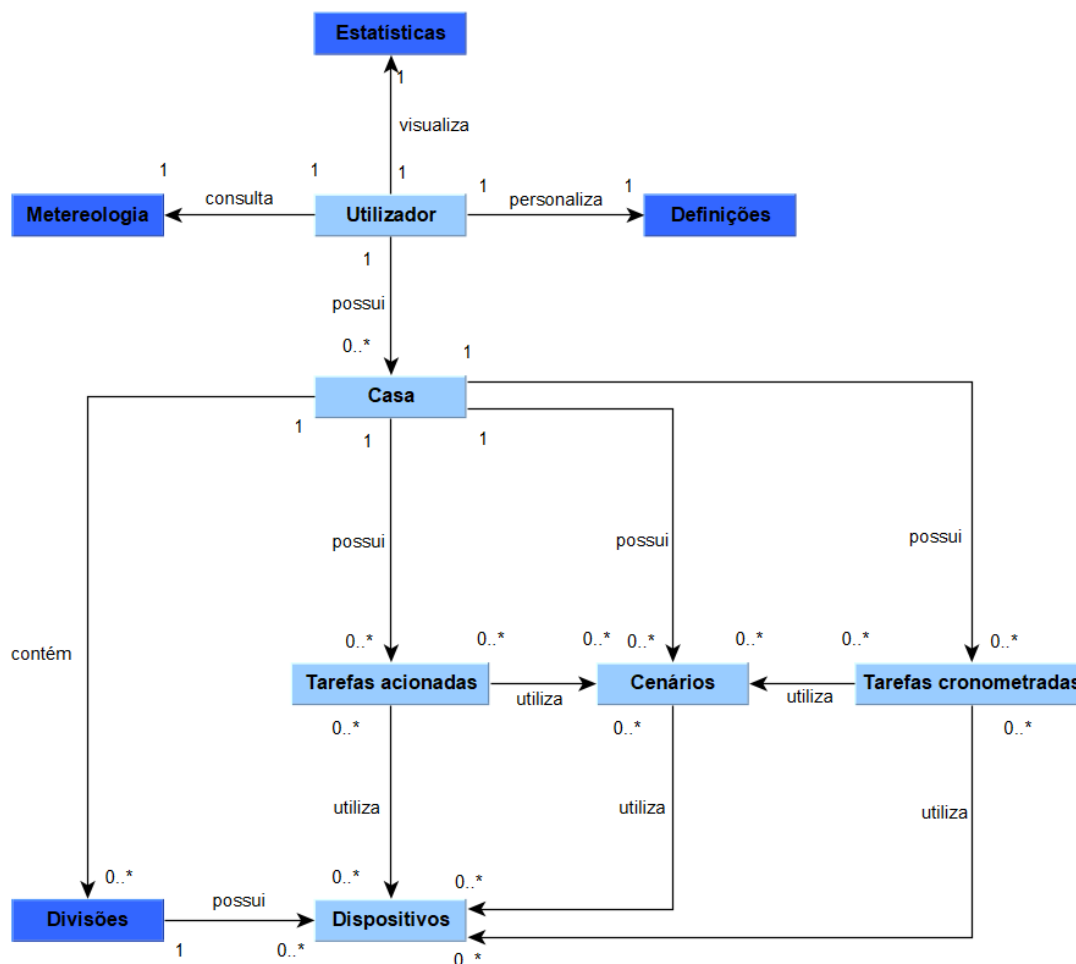


Figura 5: Diagrama de domínio do problema

Neste modelo de domínio, é possível identificar alguns aspetos:

- O utilizador possui diversas casas, sendo que cada uma destas pode ter diversas divisões e cada divisão pode possuir múltiplos dispositivos.
- Os dispositivos fazem parte dos cenários, tarefas temporais e tarefas acionadas.
- Os cenários fazem parte de tarefas temporais e tarefas acionadas.

- Cada utilizador terá as respetivas definições, dados meteorológicos e as suas estatísticas.
- Cada casa possui divisões, cenários, tarefas temporais e tarefas acionadas.

Através do modelo de domínio foi possível realizar uma primeira análise à solução que deverá ser realizada, tendo servido para identificar o problema e possíveis identidades a adicionar no sistema. Ficou assim comprovada a importância deste modelo nesta dissertação, embora que insuficiente para identificar a lógica que será necessária para desenvolver o produto.

### 3.2.2 Casos de uso

Depois da especificação dos requisitos, nomeadamente os requisitos funcionais, a conceção destes diagramas é realizada de uma forma mais facilitada. Tal como o objetivo dos requisitos funcionais é representar as suas funcionalidades, os casos de uso seguem o mesmo caminho, descrevendo o que o sistema deve fornecer, sem descrever como o deverá fazer.

Nas funcionalidades contempladas neste projeto irão constar dois tipos de atores: os utilizadores, que correspondem a qualquer consumidor da aplicação, e o sistema, que essencialmente irá executar tarefas em *background*. Pelo facto de existir um extenso leque de funcionalidades a apresentar, estas serão representadas em 4 diagramas diferentes.

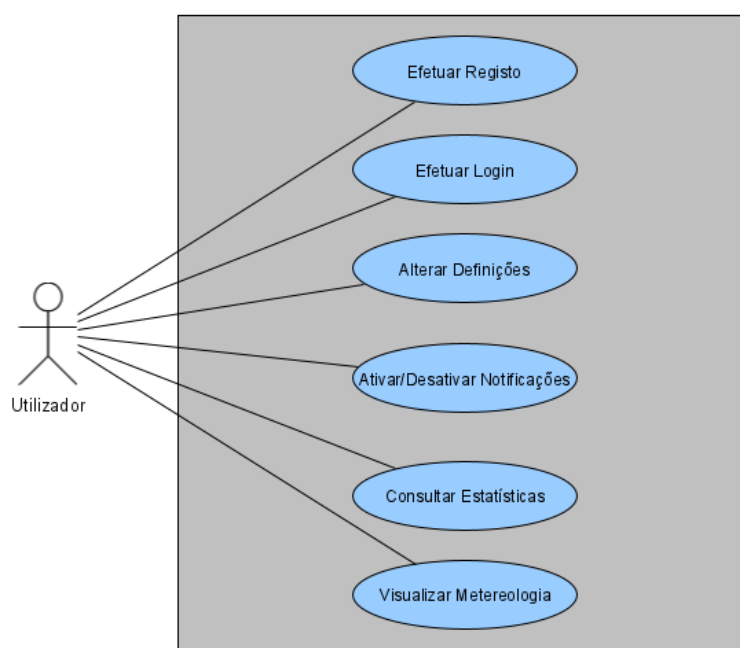


Figura 6: Casos de Uso - Gestão de Utilizadores

Numa primeira instância (Figura 6) temos os casos de uso referentes à gestão de utilizadores. Aqui são representadas as funcionalidades a que o utilizador terá acesso para manusear a sua própria informação, a começar pelo seu registo e respetivo login. Com a autenticação efetuada, o utilizador poderá alterar a sua própria informação, ativar ou desativar notificações de tarefas e outro tipo definições (como por exemplo o idioma e o tema da aplicação). Terá também acesso à meteorologia para a sua cidade pré-definida e às estatísticas das suas entidades (casas, divisões ou dispositivos).

Os utilizadores, depois de autenticados, podem também então começar a usar as funcionalidades "core" da aplicação (Figura 7), começando por adicionar as entidades que desejar. Terá primeiramente que adicionar a sua casa ao sistema e nessa mesma casa definir as suas divisões. A cada uma desta vai ser possível também adicionar os seus dispositivos, e sempre que esta tarefa for executada, outras duas serão simultaneamente processadas pelo sistema: a procura do Hub pelos dispositivos e adição de uma tarefa escalonada para que sejam atualizadas as estatísticas ao longo do tempo.

Será também possível serem adicionados cenários de utilização, que correspondem a um conjunto de dispositivos e como tal os utilizadores poderão também adicionar dispositivos a cada um destes cenários.

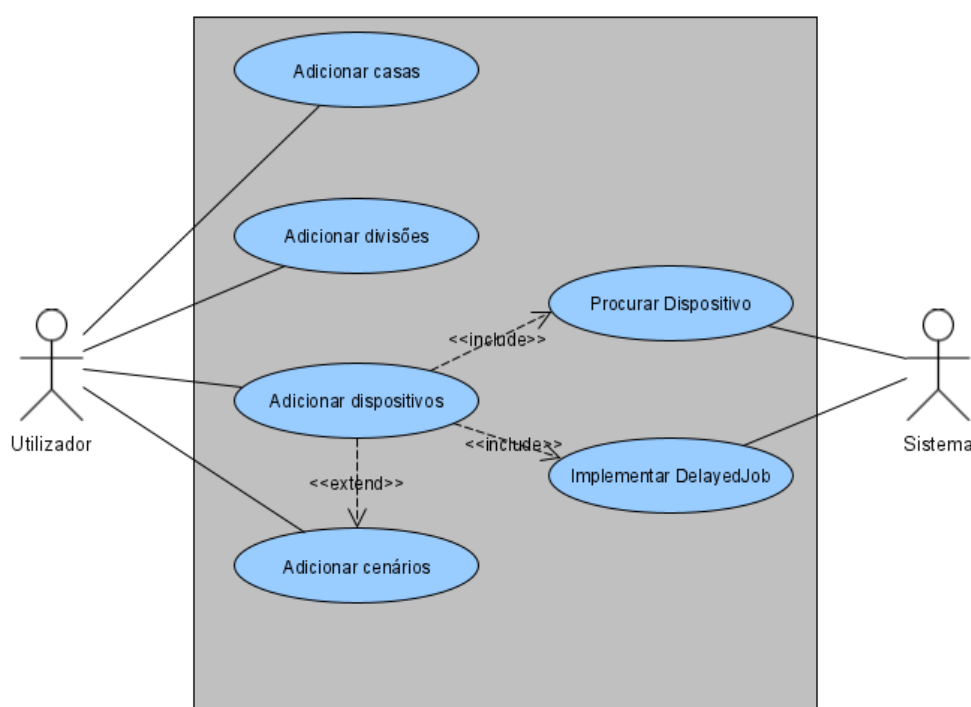


Figura 7: Casos de Uso - Adicionar entidades

É também essencial fornecer aos utilizadores a capacidade de adicionar novas tarefas (Figura 8). Existem dois tipos de tarefas que podem ser adicionadas, as temporais e as acionadas. Nas primeiras, podem selecionados



dispositivos e os respetivos estados ou então cenários, para que num determinado período do tempo sejam acionados. As tarefas acionadas podem também atribuir um dispositivo ou um cenário para serem acionados, mas sempre em conjunto com um dispositivo espoletar estas tarefas.

Sempre que cada uma destas tarefas é criada, o sistema tem a função de gerar uma tarefa escalonada, com o objetivo de em segundo plano executar estas tarefas.

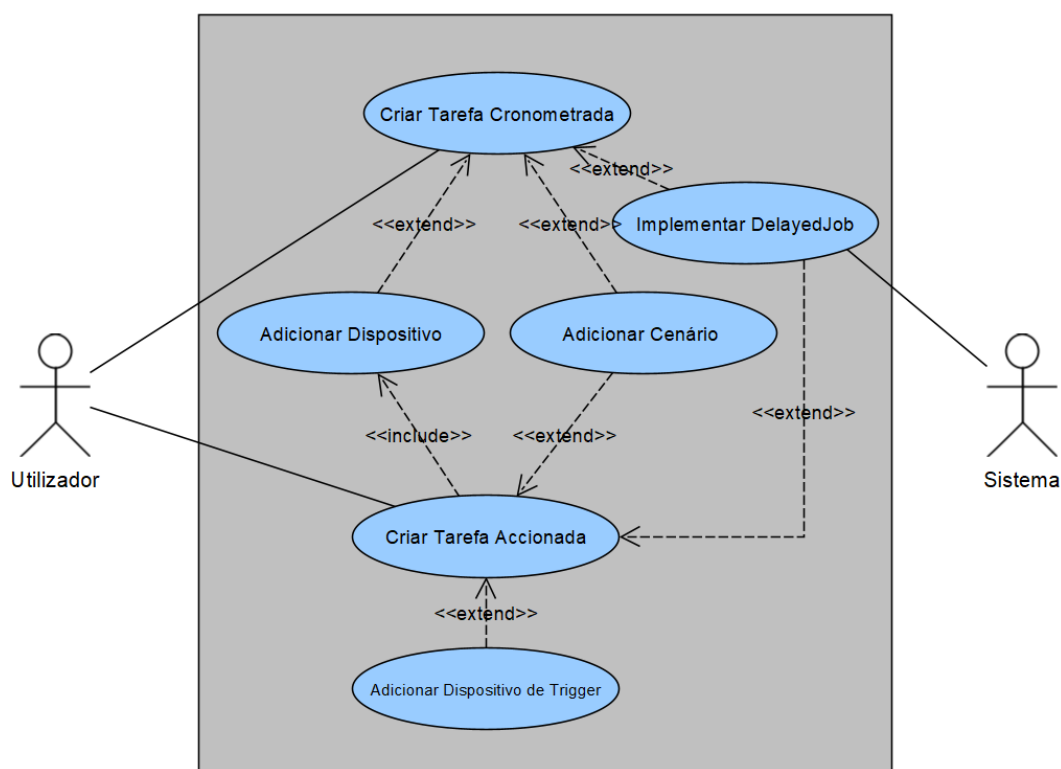


Figura 8: Casos de Uso - Criar tarefas

Por fim, e por não ser possível apresentar todas as funcionalidades que serão desenvolvidas no projeto sem tornar os diagramas extensos, na Figura 9 são resumidas as restantes tarefas em packages para cada entidade. Cada um destes packages são definidos como "Gestão" de diversas entidades dado que em cada um destes existem funcionalidades que permitem alteração informação das mesmas. Alguns exemplos são "adicionar fotografia de casa", "alterar os estados dos dispositivos", "adicionar os cenários", "editar a informação das entidades" ou "eliminar entidades".

Todos estes casos de uso estão especificados na secção A.1, onde é feita uma descrição mais específica daquilo que é esperado que seja o fluxo em cada caso de uso.

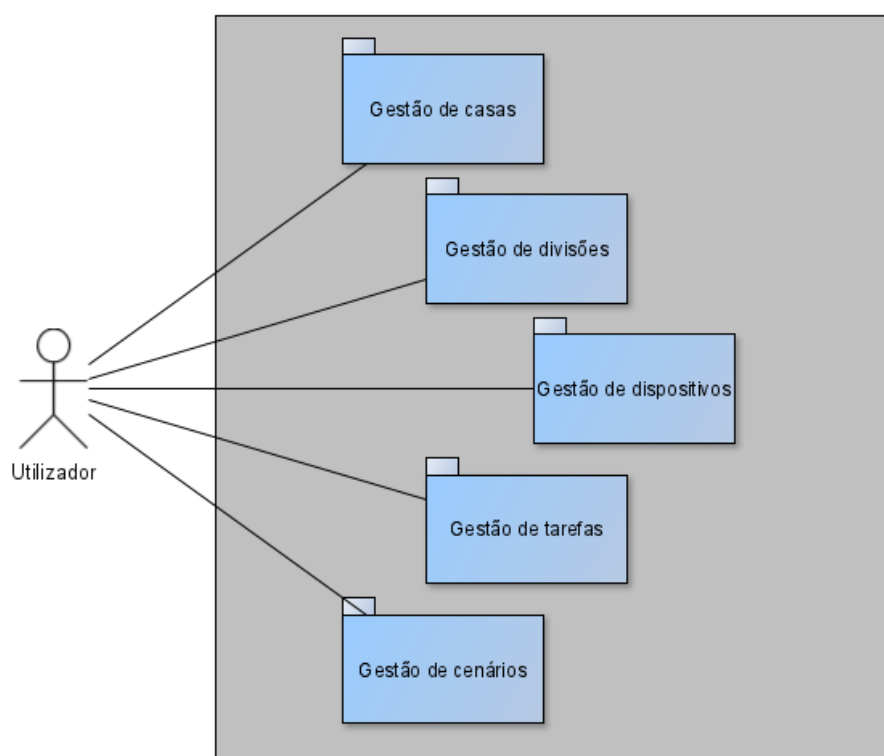


Figura 9: Casos de Uso - Gestão das entidades

### 3.2.3 Diagramas de classes

É um diagrama imprescindível e fundamental para se definir a estrutura de cada classe, tendo em consideração que as linguagens utilizadas utilizam o paradigma de Programação Orientada a Objetos. No diagrama de classes são definidos para além das classes também os atributos, métodos e relacionamentos entre as classes. Este tipo de diagrama é usado pelos arquitetos para que estes possam validar a solução previamente esboçada em outros modelos e pelos desenvolvedores para que possam implementar e documentar a solução.

A elaboração do diagrama de classes permitiu de uma forma mais eficiente definir a estrutura da arquitetura, através da representação destes elementos descritos. Foi fundamental para que se pudesse identificar, a par do modelo de domínio, todos os elementos que deveriam ser adicionados aos serviços que são consumidos pela aplicação móvel. Uma vez que o diagrama de classes completo é extenso para ser visualizado numa só imagem, será dividido em duas partes:

Numa primeira representação (Figura 10), é descrita a entidade Users que representa os utilizadores registados no sistema. Cada um destes poderá efetuar o seu registo e respetivo login no sistema. A API previamente construída não estava preparada para enviar a informação relativa à cidade do utilizador e como tal esse atributo

foi adicionado. Este campo é imprescindível para que obter informação relativa à meteorologia. A meteorologia será representada através da entidade *Weather*. Esta classe terá como atributos descrição, ícone representativo, a temperatura, humidade, hora do nascer e por do sol e a velocidade do vento. Cada utilizador terá também as respetivas Definições, representadas através da entidade *Setting*. A esta classe foram adicionados os atributos idioma, que representará o idioma da aplicação; tema, correspondente ao layout definido pelo utilizador, e notificações de tarefas acionadas e temporais que definem o envio ou não de notificações.

Outra entidade adicionada à arquitetura foi as Divisões, em que os atributos são o respetivo id, o nome, o proprietário (caso seja um quarto) e o respetivo ícone. As divisões pertencem às Casas e podem possuir múltiplos Dispositivos.

Cada Casa pode possuir ilimitados Cenários, Tarefas Temporais, Tarefas Acionadas. Nas tarefas temporais, podem ser espoletados a uma determinada hora um dispositivo ou um cenário (por exemplo ligar uma lâmpada às 22:00). Nas tarefas acionadas são também utilizados cenários ou dispositivos, mas aqui não é utilizado o tempo como acionador, mas sim um dispositivo que assume um determinado estado.

Na segunda parte do Diagrama de Classes (Figura 11) é feita uma representação do relacionamento entre algumas entidades com as Estatísticas e com os Delayed Jobs.

As estatísticas irão guardar informação de diferentes entidades, nomeadamente os Dispositivos, Tarefas Temporais, Tarefas Acionadas e os Cenários. Sempre que for acionado um cenário, por exemplo, é adicionada na Estatísticas essa mesma informação. Quanto aos Dispositivos, a cada 3 minutos será obtido o estado dos mesmos, para que se possam apresentar estatísticas de utilização.

Para que sejam executadas tarefas no *background* da aplicação, é utilizada a biblioteca *DelayedJobs* da linguagem Ruby, sendo que essas tarefas são registadas na correspondente tabela *DelayedJobs*. As tarefas temporais e acionadas têm os seus *Delayed Jobs* registados nesta tabela, a par dos dispositivos, em que para cada um deles é registado o seu estado.

O sistema é composto por muitas mais classes ao longo de toda a arquitetura, no entanto não serão descritas neste diagrama de classes por não terem o mesmo nível de relevância para o projeto. As classes aqui apresentadas foram escolhidas pela importância que as mesmas têm naquilo que é o "core" da aplicação a desenvolver, diminuindo deste modo a informação apresentada para representar apenas os conceitos essenciais.

A elaboração destes diagramas de classes tornou mais facilitado o desenho final da arquitetura, tendo possibilitado não só identificar atributos que faltavam ser acrescentados às entidades existentes, mas também construir entidades de raiz. Espera-se que estes diagramas se tornem também muito úteis na fase de desenvolvimento, reduzindo o seu tempo de desenvolvimento e prevenindo falhas nessa fase.

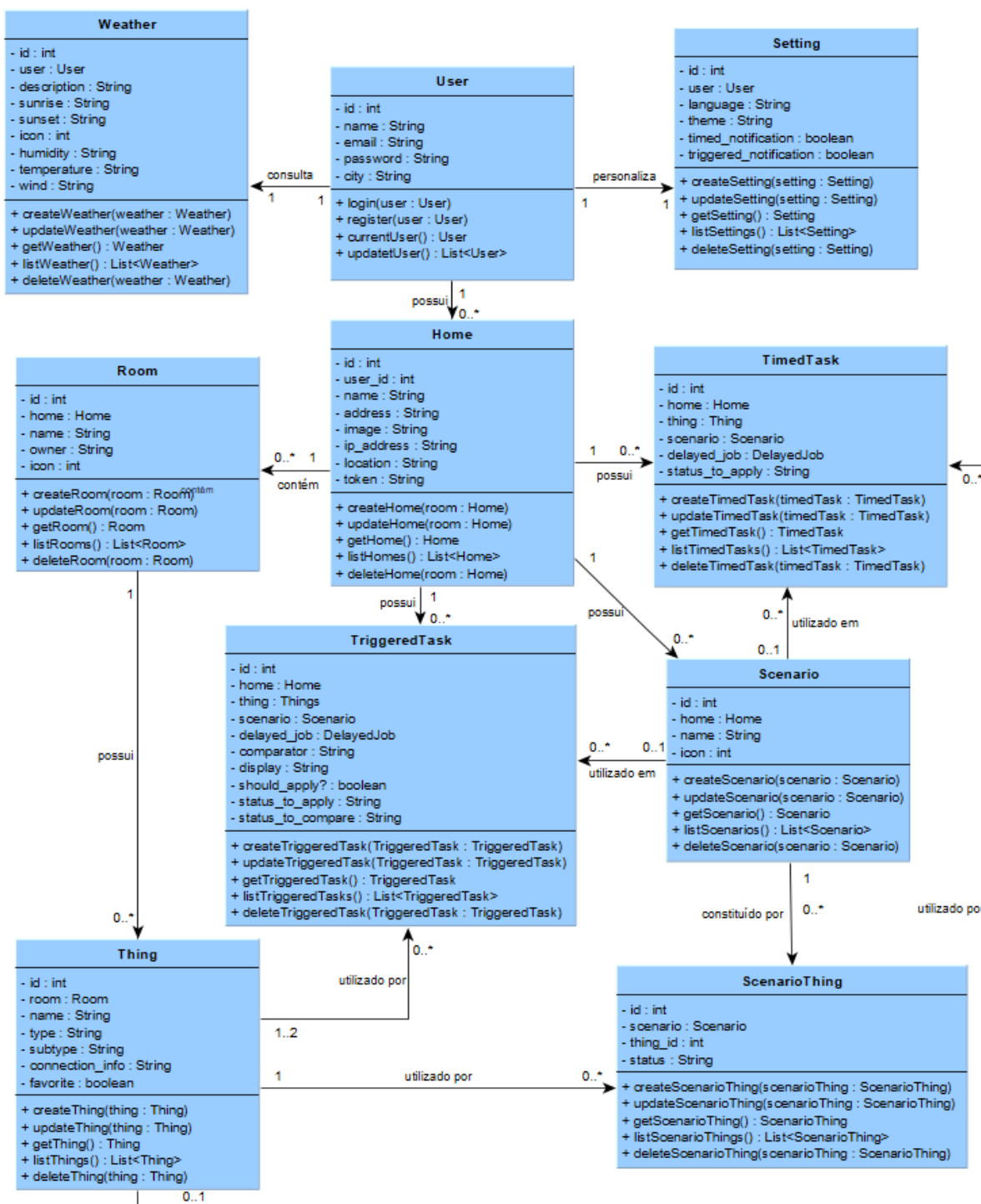


Figura 10: Diagrama de Classes - Visão Geral

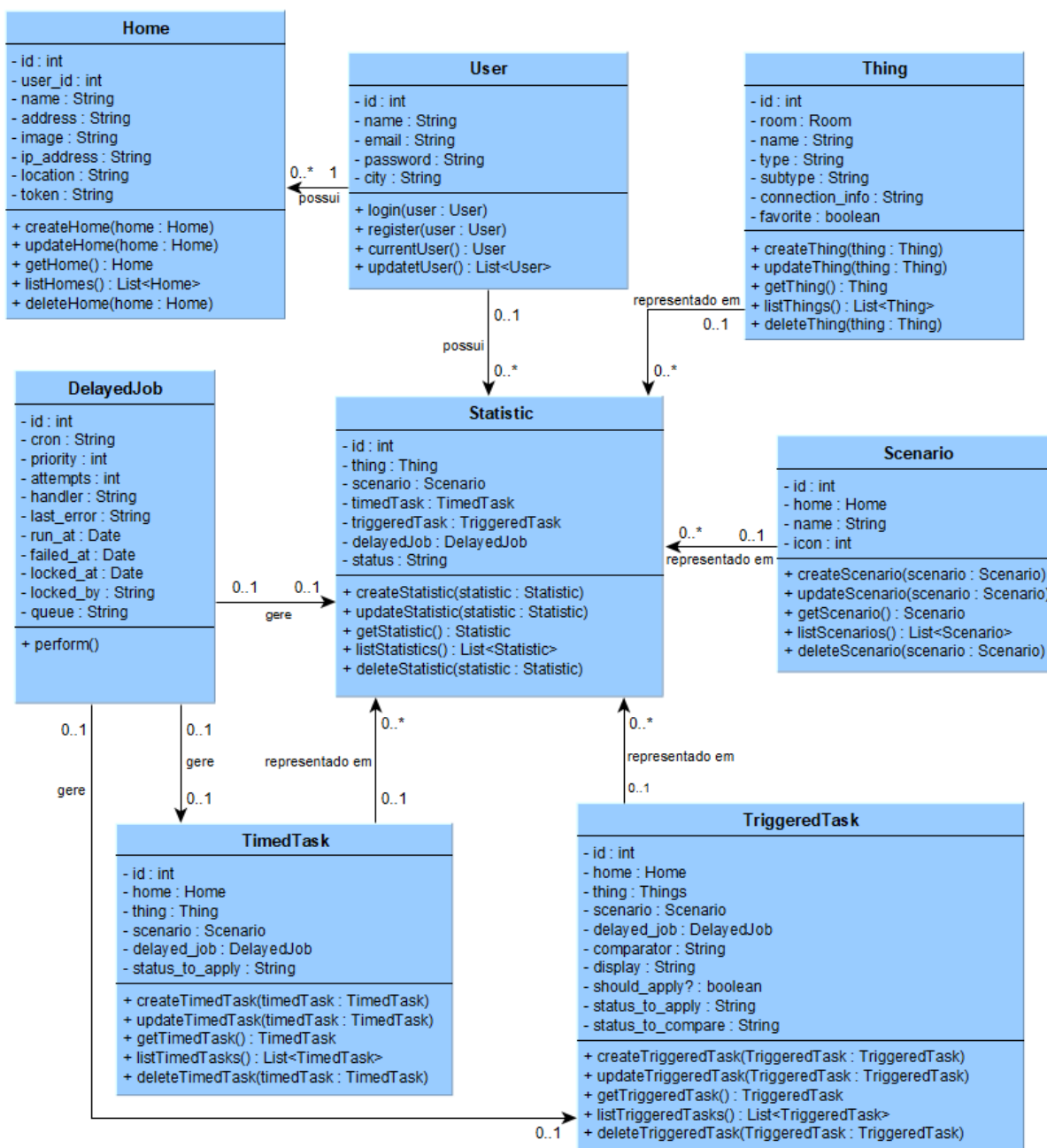


Figura 11: Diagrama de Classes - Estatísticas e Delayed Jobs

---

## IMPLEMENTAÇÃO, TESTES E DEPLOYMENT

---

Após o estudo dos diferentes conceitos e tecnologias inerentes a esta dissertação e da análise e concepção da solução, as próximas etapas a seguir no processo de desenvolvimento de um software são as seguintes:

- **Implementação:** É nesta fase onde é aplicado todo o estudo adquirido e se transformam os diagramas desenhados em código-fonte, tendo em consideração que é necessário satisfazer os requisitos levantados e a respetiva priorização. Durante esta etapa são descritas todas as tecnologias utilizadas, os desenvolvimentos e a solução concebida.
- **Testes:** Após implementar a solução, é necessário testar o que foi desenvolvido e garantir que as funcionalidades estão corretamente implementadas.
- **Deployment:** Numa última fase deste capítulo, a aplicação deverá ser disponibilizada aos utilizadores. Para isso será necessário colocar a aplicação num servidor pronta a poder ser descarregada.

### 4.1 FERRAMENTAS E TECNOLOGIAS

Nesta secção serão documentadas todas as ferramentas que foram aplicadas nesta dissertação para que fosse possível satisfazer os requisitos definidos e os modelos desenhados.

#### 4.1.1 *Seleção da Framework Híbrida*

Numa primeira instância da fase de implementação, depois de ser efetuado o estudo das tecnologias e serem elaborados diferentes diagramas, é necessário efetuar decisões. A primeira e talvez das mais importantes desta dissertação é a escolha da framework que será utilizada para o desenvolvimento de aplicações multiplataforma. Foram estudadas duas frameworks que permitem o desenvolvimento de aplicações híbridas: React Native e Ionic Framework. A análise efetuada permitiu aferir que a melhor opção seria a utilização de Ionic Framework. O principal fator de decisão foi a curva de aprendizagem que diferencia as tecnologias: o tempo necessário para que

sejam desenvolvidas aplicações através da utilização de React Native é superior a Ionic, até porque nesta última, as tecnologias utilizadas durante o desenvolvimento (JavaScript, CSS e HTML) são tecnologias anteriormente utilizadas noutros projetos, havendo assim já alguma familiaridade com as mesmas.

Outra característica que favorece a utilização de Ionic em detrimento de React Native é a forma como a componente gráfica da aplicação é realizada. Na framework Ionic, é utilizado HTML para definir o esquema das interfaces gráficas, enquanto que em React Native estas interfaces são construídas através de JSX, funções JavaScript que permitem a inserção de HTML. Este nível de flexibilidade que React Native fornece, é possível atingir de igual modo com Ionic através da utilização de outras linguagens, como por exemplo Angular.

A linguagem de programação utilizada em React Native é JavaScript, mas todos os componentes utilizados são iOS e Android, permitindo desta forma o desenvolvimento híbrido. Ionic Framework, por outro modo, só poderá atingir esta característica através da utilização de outras tecnologias, como o Apache Cordova. Esta divergência leva a que ao desempenho na execução das funcionalidades, principalmente quando se recorre às API's nativas, seja ligeiramente inferior através da utilização de Ionic, tornando-o assim mais lento que React Native.

Poderá ser perdida algum desempenho ao não optar pela escolha de React Native, contudo é preferível optar pela utilização de Ionic Framework, e proporcionar ao utilizador uma interface mais elegante e reduzir o tempo de desenvolvimento.

Depois de analisadas e comparadas as duas tecnologias, a seleção da framework híbrida a utilizar recaiu em Ionic Framework por garantir um menor tempo de desenvolvimento e pela possibilidade de proporcionar interfaces gráficas mais agradáveis ao utilizador.

#### 4.1.2 Node.js

O Node.js apareceu em 2010 e é uma plataforma *open-source* que fornece ferramentas para o desenvolvimento de aplicações JavaScript, ferramentas estas que são executadas do lado do servidor.

A sua utilização alterou o paradigma do desenvolvimento em JavaScript: os programadores passaram de desenvolver código que apenas poderia ser executado em browsers, para terem a possibilidade de executar o código nas próprias máquinas através de aplicações standalone, deixando assim a linguagem JavaScript de representar apenas uma linguagem para desenvolvimento de aplicações Web interativas.

*Node Package Manager*, ou *npm*, é o repositório do Node.js, onde é possível encontrar diversas bibliotecas, que podem ser utilizadas para o desenvolvimento de aplicações JavaScript. A sigla *npm* corresponde também ao comando que pode ser executado, seguido do nome da biblioteca, de modo a proceder à instalação destes pacotes em projetos de um modo bastante simplificado.

Nesta dissertação foram utilizadas algumas bibliotecas, das quais três se destacam pela sua importância nas funcionalidades da aplicação: a primeira e que apresenta uma maior importância, foi a biblioteca que fornece as

ferramentas de desenvolvimento em *Ionic Framework* <sup>1</sup>. Foi também necessário integrar uma biblioteca para a tradução de todo o texto da aplicação, mediante o idioma preferencial, designada *Translate* <sup>2</sup>. Por fim, foi utilizada também uma biblioteca que, entre outras funcionalidades, permite a conversão das datas em diferentes formatos, designada *moment* <sup>3</sup>.

#### 4.1.3 OpenWeatherMap

OpenWeatherMap é um serviço online que fornece informação das condições meteorológicas, previsões e históricos para milhares de cidades ao longo do globo. Este serviço possui uma API, o que possibilita a invocação dos seus serviços e da respetiva informação por parte dos desenvolvedores nas suas aplicações.

Para que seja possível a utilização desta API, é necessário numa primeira instância, que seja realizado o registo através da sua página <sup>4</sup> e após esse registo, esperar algumas horas até que a conta seja ativada. Após a validação da mesma, o API ID (chave identificadora da conta) fica disponível para efetuar chamadas ao serviço.

Este serviço possui uma documentação completa onde são indicados vários exemplos de pedidos GET que podem ser efetuados de modo a obter informação. Por exemplo, se for pretendido obter as condições meteorológicas para Lisboa, o URL seria construído desta forma:

```
http : //api.openweathermap.org/data/2.5/weather?q = Lisbon,pt&APPID = {APIKEY}
```

Este pedido efetuado resulta numa resposta no formato json, com toda a informação sobre as condições meteorológicas para a cidade (Figura 12).

---

1 <https://www.npmjs.com/package/@ionic/cli>

2 <https://www.npmjs.com/package/translate>

3 <https://www.npmjs.com/package/moment>

4 <https://openweathermap.org>



```
{
  "coord": {
    "lon": -9.13,
    "lat": 38.72
  },
  "weather": [
    {
      "id": 801,
      "main": "Clouds",
      "description": "céu pouco nublado",
      "icon": "02d"
    }
  ],
  "base": "stations",
  "main": {
    "temp": 17.22,
    "feels_like": 14.66,
    "temp_min": 16.11,
    "temp_max": 18.89,
    "pressure": 1010,
    "humidity": 72
  },
  (...),
  "visibility": 10000,
  "wind": {
    "speed": 4.6,
    "deg": 130
  },
  "clouds": {
    "all": 20
  },
  "dt": 1586013119,
  "sys": {
    "type": 1,
    "id": 6901,
    "country": "PT",
    "sunrise": 1585980958,
    "sunset": 1586026946
  },
  "timezone": 3600,
  "id": 2267057,
  "name": "Lisboa",
  "cod": 200
}
```

Figura 12: Resposta de um pedido realizado ao openWeatherMap

No âmbito desta dissertação, esta API será utilizada para obter as condições meteorológicas para a cidade que o utilizador irá configurar. Essas condições meteorológicas são a temperatura, humidade, horas de nascer do sol e pôr do sol, ícone e descrição. Utilizaremos o plano *free*, e como tal o número de invocações à API num determinado período de tempo é limitado. Para evitar que sejam efetuadas inúmeras chamadas à API, foi aplicado um tempo mínimo de seis minutos antes de voltar a invocar o serviço.

#### 4.1.4 OneSignal

Uma das características que uma grande parte das aplicações móveis apresentam é a capacidade de enviar notificações *push*. Este tipo de notificações permitem o envio de alertas ou avisos, mesmo não estando a utilizar ativamente a aplicação, que são apresentados por exemplo no ecrã de bloqueio de um dispositivo.

OneSignal é um serviço que permite o envio de notificações *push* de forma gratuita, e pode ser facilmente integrado em aplicações. Para que seja possível usufruir deste serviço, é necessário efetuar o registo de uma conta, e seguidamente configurar a aplicação. Para configurar a aplicação será necessário a utilização de outra ferramenta, Firebase<sup>5</sup>. A utilização de Firebase garante que seja gerada uma *key* da Google Server API, permitindo assim que o OneSignal utilize os serviços da Google para o envio de notificações.

Após a configuração da aplicação sobre a qual serão enviadas as notificações, o envio de notificações pode ser realizado de dois modos distintos: através do website ou através de uma outra aplicação que irá utilizar uma API para enviar notificações. No âmbito desta dissertação, serão enviadas mensagens no momento em que as

5 <https://firebase.google.com>

tarefas são espoletadas, e como tal será utilizada a API do OneSignal para o envio automático das notificações pelo backend do projeto. Toda a lógica implementada para o envio de notificações será retratada mais à frente.

## 4.2 API DO MIDDLEWARE

O middleware já descrito anteriormente corresponde ao backend do sistema, sendo que a aplicação híbrida a desenvolver no âmbito desta dissertação irá funcionar como um frontend. A aplicação, através da utilização do *wrapper*, irá consumir os métodos que interagem com a API e controlar os diferentes dispositivos.

### 4.2.1 Extensibilidade do Middleware

Apesar deste middleware apresentar uma arquitetura bem estruturada e oferecer uma API para que os seus serviços sejam consumidos por qualquer aplicação para a gestão de uma Smart Home, existem algumas limitações para que os requisitos definidos sejam satisfeitos.

Apesar de não ser o foco desta dissertação, olhando para os requisitos previamente descritos, rapidamente se percebeu que seria necessário adicionar novos componentes que seriam fundamentais no sistema para cooperar com outras entidades já existentes: *divisões*, *estatísticas*, *condições meteorológicas* e *definições*. Deste modo, tiveram de ser aplicadas alterações a nível de dois componentes importantes: a API que possui grande parte da lógica do middleware, e o Wrapper da API que possui as funções de tradução para invocar os métodos HTTP que a API oferece.

Em relação às alterações efetuadas no Hub, a ideia passou por tentar ao máximo adaptar a lógica já existente para o desenvolvimento das alterações, de modo a minimizar riscos de desenvolvimento por parte de outro desenvolvedor. Assim sendo, foram construídos numa primeira fase os *models* que representam cada um destes elementos. De seguida (4.1) é representado um *model* que representa as definições dos utilizadores, onde é feita uma validação aos atributos mandatórios (linha 2) e é definida a relação entre o utilização e a definição (linha 4).

```

1 class Setting < ApplicationRecord
2   validates :user_id, :timed_tasks_notification, :triggered_tasks_notification, :
      theme, :language, presence: true
3
4   belongs_to :user
5 end

```

Listagem 4.1: Hub - Representação de um model

Foi também necessário desenvolver os *controllers* para estes elementos, respeitando a lógica de construção de métodos para as operações CRUD. Na listagem (4.2) é apresentado um exemplo de um método de criação de uma nova divisão, onde é obtido a casa do utilizador em questão através do id recebido por parâmetro (linha 5) e é

criada a divisão com a informação recebida (linha 6). Caso este processo termine em erro, é criada uma status de *ERROR* (linha 10):

```

1 class RoomsController < ApplicationController
2   before_action :authenticate_user
3
4   def create
5     home = current_user.homes.find(params[:home_id])
6     room = home.rooms.build(room_params)
7     if room.save
8       render json: room, status: :created
9     else
10      render json: room.errors, status: :unprocessable_entity
11    end
12  end
13 end

```

Listagem 4.2: Hub - Representação de um controller

Foram adicionados também novos atributos a algumas classes já existentes, como por exemplo adicionar o atributo *image* na entidade *home*, e o atributo *icon* na entidade *scenario*. Alguns relacionamentos entre diferentes entidades tiveram de ser adicionados: os dispositivos deixaram de pertencer às casas, para fazerem parte das divisões.

É esperado que a aplicação móvel consiga apresentar informação sobre as estatísticas para dispositivos, cenários e tarefas, e como tal foi também necessário adaptar o sistema para que isto fosse possível. Foi assim desenvolvido um serviço que permite que sejam criadas estatísticas em 3 momentos distintos:

- Quando os cenários são aplicados;
- Quando as tarefas são executadas;
- Depois de ser obtido a cada 3 minutos o estado dos dispositivos.

Ao contrário dos cenários e das tarefas, no caso das estatísticas não é gerada imediatamente uma estatística, mas sim é gerado uma tarefa assíncrona que a cada 3 minutos obtém o estado de cada dispositivo, e regista-o. Este intervalo de tempo foi definido para que a base de dados não fosse sobrecarregada, mas garantindo que a informação se mantinha fidedigna. De seguida é apresentado na listagem 4.3 parte da classe que possui a lógica para a criação de estatísticas, classe genérica que permite armazenar estatísticas de dispositivos, tarefas ou cenários, e como tal possui um construtor que recebe o utilizador ao qual pertence a estatística e os parâmetros a armazenar (linha 2). Existem também os métodos *performStat* (linha 8) e *create\_stat* (linha 17) que são invocados durante as tarefas assíncronas e tratam criar e persistir a informação sobre as mesmas de forma a gerar estatísticas:

```

1 class CreateStatistic
2   def initialize(user:, params:)

```

```

3     @user = user
4     @params = params.clone
5     @status = false
6 end
7
8 def performStat
9     @statistic = user.statistics.build(params)
10    ActiveRecord::Base.transaction do
11        create_stat
12        raise ActiveRecord::Rollback if statistic.errors.count.positive?
13    end
14    statistic
15 end
16
17 def create_stat
18     return unless statistic.save
19     @status = statistic.save
20 end
21 end

```

Listagem 4.3: Hub - Criação de uma estatística

Juntamente com todas estas alterações efetuadas no Hub, foi também necessário preparar o wrapper para executar traduções a novos métodos da API. Deste modo, foram adicionadas as classes para cada endpoint criado na API, nomeadamente para as divisões, estatísticas, condições meteorológicas e definições, possibilitando assim a sua invocação na aplicação a desenvolver. Na listagem 4.4 é apresentado um exemplo de parte de uma das classes desenvolvidas, neste caso uma Divisão, onde é feita uma invocação à API de modo a criar a divisão numa casa com os parâmetros recebidos (linha 15) :

```

1 class Rooms {
2
3     constructor(axios, home) {
4         this.axios = axios;
5         this.home = home;
6     }
7
8     /**
9     * Creates a room
10    * @param {Object} room
11    * @param {string} room.name
12    * @return {Promise}
13    */
14    createRoom(room) {
15        return this.axios.post(`/homes/${this.home.id}/rooms`, { room });

```

```

16     }
17 }

```

Listagem 4.4: Representação do Wrapper

Nesta secção não foram mencionadas as alterações realizadas no middleware para que a aplicação móvel pudesse usufruir de notificações push, nem as alterações referentes ao de chaves de APIs pelo middleware para a aplicação móvel, sendo que este desenvolvimento serão descritos mais à frente, na respetiva descrição das funcionalidades.

#### 4.2.2 Utilização da API

Após serem realizadas todas as alterações na lógica já existente, a API pode ser invocada nas aplicações do lado do cliente. A sua utilização só é possível depois de ser adicionado o módulo do wrapper nas dependências do projeto.

A classe principal da API pode ser instanciada com dois argumentos, o *url*, uma String que representa o endpoint da API e um valor *boolean* que representa a *cache*, cujo objetivo é ativar o *caching* dos pedidos GET realizados à API. Para que seja possível realizar pedidos, é necessário numa primeira instância, preencher o atributo *auth*, que contém um *token* de autenticação devolvido pelo método *login* da API.

Todos os métodos desenvolvidos no wrapper da API devolvem *Promises*, um mecanismo que permite a realização de operações assíncronas, ou por outras palavras, tarefas que serão completadas no futuro, enquanto outras continuam a ser executadas, mesmo sem ainda ter existindo uma resposta. Deste modo, a utilização do mecanismo *async/await* será imprescindível nas aplicações móveis para o tratamento destas invocações assíncronas.

No seguinte excerto de código (listagem 4.5) é exemplificado o modo como são invocados estes serviços através de uma aplicação em JavaScript, onde existe uma função é inicialmente criado um objeto Utilizador (linha 3), é realizada a sua autenticação no sistema (linha 10/linha 11) e são obtidas as casas desse mesmo utilizador (linha 13):

```

1  function getHomes() {
2
3      const user = {
4          name: "Leonel",
5          email: "leonel@gmail.com",
6          city: "Lisboa",
7          password: "qwerty"
8      };
9
10     let currentUser = await homewatch.users.login(user);
11     homewatch.auth = currentUser.data.jwt;
12

```

```

13     let homes = await homewatch.homes.listHomes();
14 }

```

Listagem 4.5: Exemplo de utilização do Wrapper

### 4.3 DESCRIÇÃO DA IMPLEMENTAÇÃO

Nesta secção pretende-se apresentar com um maior detalhe parte das estratégias adotadas no desenvolvimento da aplicação desenvolvida nesta dissertação, como por exemplo o consumo de APIs nativas, a utilização de componentes Ionic e customizados e os mecanismos para envio de notificações.

#### 4.3.1 Estrutura da Aplicação

A aplicação foi estruturada para que exista uma manutenção facilitada e um reaproveitamento do código para diferentes funcionalidades. Foi também imperativo tornar a estrutura e o código perceptível de modo a que diferentes programadores consigam entender o mesmo com o mínimo esforço exigido.

Após o estudo realizado à framework utilizada no projeto a estrutura implementada na aplicação foi a seguinte:

**[node\_modules/]** - contém todos os módulos necessários no projeto.

**[src/]** - contém o código-fonte da aplicação.

**[app/]** - possui toda a informação para iniciar e estruturar a aplicação.

**[assets/]** - aqui são armazenando recursos, como por exemplo as imagens.

**[helpers/]** - aqui são desenvolvidos métodos genéricos que deverão ser utilizados ao longo do projeto.

**[pages/]** - contém o código-fonte das funcionalidades, subdividido em diferentes componentes.

**[homes/]** - contém a lógica para listar e criar casas.

**[rooms/]** - contém a lógica para listar e criar divisões.

**[things/]** - contém a lógica para listar e criar dispositivos.

**[scenarios/]** - contém a lógica para listar e criar cenários.

**[scenario\_things/]** - contém a lógica para adicionar dispositivos nos cenários.

**[statistics/]** - contém a lógica para listar as estatísticas para as casas, dispositivos, cenários e tarefas.

**[tasks/]** - contém a lógica para listar e criar tarefas.

**[users/]** - contém a lógica para registar/autenticar utilizadores e apresentar a homepage.

**[services/]** - contém a lógica para utilização de serviços externos.

**[package.json]** - ficheiro que contém todas as dependências do projeto.

**[tsconfig.json]** - ficheiro que indica os *root files* e contém as opções de compilação.

**[tslint.json]** - ficheiro que contém as regras para o comportamento do TypeScript.

**[README.md]** - ficheiro que contém a informação essencial sobre o projeto.

#### 4.3.2 Consumo de APIs nativas

Como já visto anteriormente, Ionic Framework possui a capacidade de utilização de SDK's nativos e assim reunir uma interface sólida e fluida com uma bom desempenho na utilização de funcionalidades nativas dos dispositivos. Através da integração de Cordova é possível fazer deploy como uma aplicação nativa ou então correr a aplicação via browser, como uma **PWA**.

Analisar o comportamento das APIs nativas através de uma aplicação híbrida é um ponto fundamental no âmbito desta dissertação, e como tal recorreu-se a algumas das bibliotecas suportadas pela framework. Neste projeto foram utilizadas três delas: *câmara*, *geolocalização* e *vibração*.

##### Câmara

A API da câmara foi utilizada para possibilitar que o utilizador represente as suas casas com imagens à sua escolha. Pode ser utilizado de duas formas: aceder à galeria e fazer o carregamento de uma imagem ou então tirar uma fotografia no momento.

Uma boa prática para o consumo de componentes reutilizáveis é a criação de um *helper*, onde são desenvolvidos métodos que poderão ser consumidos variadas vezes. Nestes métodos é enviado para as funções da biblioteca um conjunto de opções que irão definir de que forma será consumida a API nativa, que neste caso poderá ser para realizar o upload de imagens ou para tirar fotografias. Na listagem 4.6 é representada a lógica para a utilização da câmara do dispositivo do utilizador, onde inicialmente criada uma instância de um objeto *Camera* (linha 3) e configuradas as opções que a mesma deverá assumir (linha 12), para que posteriormente seja tirada a fotografia através da câmara do dispositivo (linha 19) .

```
1     function getPicture() {
2
3     let camera = new Camera();
4
5     /**
6     * Parameters
7     * @param {quality} - Picture quality in range 0-100.
8     * @param {destinationType} - Choose the format of the return value.
9     * @param {encodingType} - Choose the returned image file's encoding.
10    * @param {mediaType} - Set the type of media to select from.
11    */
12    const options: CameraOptions = {
13        quality: 70,
14        destinationType: camera.DestinationType.DATA_URL,
15        encodingType: camera.EncodingType.PNG,
```

```

16     mediaType: camera.MediaType.PICTURE
17   };
18
19   let imageData = await camera.getPicture(options);
20
21   return `data:image/png;base64,` + imageData;
22 }

```

Listagem 4.6: Utilização da API da Câmara

### Geolocalização

A utilização desta biblioteca nesta aplicação é um recurso importante para obtenção da localização do utilizador e das condições meteorológicas para a respetiva região. Esta biblioteca, após o utilizador aceitar a permissão imposta pelo seu dispositivo, consegue aceder à API nativa e recorrer ao Global Positioning System (GPS) para obter informações como a latitude e longitude do dispositivo.

Da mesma forma que a lógica desenvolvida para a API da Câmara, aqui foi também desenvolvido um *helper* que possui um método que recorre à API para obter em real time as coordenadas do dispositivo. Com estas coordenadas, foi utilizada uma outra biblioteca que realiza a "tradução" das coordenadas para a cidade em questão. No pedaço de código presente na listagem 4.7 é representada a função *getLocation()* desse *helper* que trata de invocar a API nativa de geolocalização (linha 5), depois de ter sido criado um objeto do tipo *Geolocation* (linha 3).

```

1  function getLocation()
2  {
3    let geolocation = new Geolocation();
4
5    geolocation.getCurrentPosition().then((resp) => {
6      return geolocation;
7    }).catch((error) => {
8      console.log('Error getting location', error);
9    });
10
11   return ;
12 }

```

Listagem 4.7: Utilização da API da Geolocalização

### Vibração

Um recurso que promove a intuição na utilização de aplicações é a vibração do dispositivo para indicar que um processo foi executado. Neste projeto, devido ao facto do processo de pesquisa de equipamentos IoT decorrer



entre 5-10 segundos, foi pensado em adicionar esta feature no momento em que o dispositivo é encontrado, de forma a captar novamente a atenção do utilizador.

Tal como para as duas outras bibliotecas, foi também desenvolvido um *helper* específico para a invocação do método de vibração, até por ser um recurso que possivelmente será no futuro integrado em outras funcionalidades. Na listagem 4.8 é representada a função *vibrate()* desse *helper* que trata de invocar a API nativa para que seja efetuada a vibração do dispositivo (linha 4).

```
1  function vibrate()
2  {
3    let vibration = new Vibration();
4    vibration.vibrate(1000);
5  }
```

Listagem 4.8: Utilização da API da Vibração

#### 4.3.3 Componentes Ionic

Como já foi visto anteriormente, as aplicações desenvolvidas através desta framework são construídas a partir de blocos reutilizáveis de código, designados de componentes, que permitem um rápido desenvolvimento da *User Interface* das aplicações. Ionic fornece uma extensa lista dos seus próprios componentes como por exemplo tabs de navegação, botões, menus ou ícones. Estes componentes apesar de serem pré-construídos pela framework, podem ser customizados de forma a serem adaptados ao resto da interface da aplicação.

No âmbito desta dissertação, estes componentes foram utilizados por dois motivos: para que possa ser estudado o desempenho e usabilidade destes componentes numa solução para smart homes e também por a reutilização destes componentes diminuir o esforço despendido no desenvolvimento da aplicação.

#### 4.3.4 Integração com Angular

Depois de ser representada a integração de Ionic com Cordova, a framework que possibilita o acesso às APIs nativas dos dispositivos, é a vez de ilustrar a integração de Ionic Framework com Angular. A utilização de Angular foi imprescindível neste projeto principalmente pelo recurso de Data-Binding que esta framework oferece. Este recurso garante uma sincronização automática entre a parte lógica da aplicação e a view. Assim, sempre que uma variável atualiza o seu valor, este é imediatamente alterado na interface, sem que o backend da aplicação tenha de executar processos extra. O acesso por parte das views às variáveis dos componentes, com a utilização deste recurso, é feita através da utilização de *chavetas* no código HTML, como representado abaixo:

```
<ion-row >
  <h1> {{roomsLabel}} </h1>
```

```
<button (click)="newRoom(home)"> {{addRoomLabel}} </button>
</ion-row>
```

Listagem 4.9: Exemplo da utilização de Angular

No excerto de código da listagem 4.9 está também representado um recurso importante que esta framework oferece: os eventos espoletados pelo utilizador, que são adicionados na view através da utilização da keyword *click*, juntamente com a indicação da função que irá ser espoletada através do clique.

Outro recurso que Angular forneceu na sua integração com Ionic foram as diretivas estruturais, marcadores que informam qual o comportamento que um elemento deverá tomar. Algumas destas diretivas podem alterar completamente o layout do componente, podendo adicionar ou remover elementos na página. Em baixo estão representados algumas das diretivas utilizadas com respetivos exemplos:

- **ngIf** - é utilizado para definir se um elemento deverá ou não ser renderizado.

```
<div *ngIf="loading">
  
</div>
```

- **ngFor** - processa cada item de um objeto iterável, adicionando o mesmo número de itens que o tamanho desse mesmo objeto.

```
<div *ngFor="let thing of scenarioThings">
  <div> {{thing.thing.name}} </div>
  <div> {{thing.thing.room.name}} </div>
</div>
```

- **ngClass** - permite definir dinamicamente a classe de um elemento.

```
<ion-content [ngClass]="{'homepage': theme == 'homepageDefaultTheme', '
homepageRedTheme': theme=='red' , 'homepageGreenTheme': theme=='green' }">
```

- **ngModel** - vincula-se a uma variável onde é armazenado o seu valor.

```
<ion-segment [(ngModel)]="tasks_type" (ngModelChange)="onTypeChange($event)
">
  <ion-segment-button value="timed">
    {{timedLabel}}
  </ion-segment-button>
  <ion-segment-button value="triggered">
    {{triggeredLabel}}
```

```
</ion-segment-button>
</ion-segment>
```

Angular tem a capacidade de gerir eventos de forma bastante efetiva ao longo da aplicação, em diferentes páginas. Este recurso segue um padrão *observable* onde é numa primeira instância definido o evento que irá espoletar uma mudança de estado (*publish*) e seguidamente definido o serviço que irá depender dessa execução (*subscribe*). No excerto de código da listagem 4.10 é apresentado um exemplo da utilização deste paradigma, que representa um evento de mudança de estado do sensor de um dispositivo.

```
async onStatusChange() {
  this.events.publish(`thing:status:update:out${this.thing.id}`, this.status);
}

await this.events.subscribe(`thing:status:update:out${thing.id}`, newStatus => {
  this.onStatusToApplyChange(newStatus, 'thing');
});
```

Listagem 4.10: Exemplo da aplicação do padrão Observable

Estes exemplos descritos não representam todas as utilizações da framework na implementação do projeto, mas sim as mais impactantes na aplicação, até porque teve de ser realizada uma seleção de modo a não tornar esta secção demasiado extensa.

#### 4.3.5 Gráficos

Um dos requisitos definidos na fase inicial do projeto foi a possibilidade de o utilizador manter uma gestão de custos dos seus dispositivos e serem apresentadas estatísticas de utilização dos mesmos. A forma mais efetiva de fornecer estatísticas aos utilizadores é a apresentação de diversos gráficos, tornando assim mais agradável a análise realizada pelo utilizador.

No contexto de representação de gráficos em aplicações móveis desenvolvidas em JavaScript destaca-se a utilização de uma biblioteca designada *Chart.js*, onde a sua utilização torna o processo de inserção de *data* para produção dos gráficos facilitado. A biblioteca usufrui do elemento do HTML5 *<canvas>* para produção dos gráficos, o que é perfeito para as aplicações que utilizam esta linguagem, que é o caso das aplicações híbridas. Para que seja possível a utilização desta biblioteca na aplicação, é necessária a integração de Angular com Ionic, integração esta que já foi descrita na secção anterior.

Neste projeto estes gráficos são utilizados em 3 situações distintas: horas do dia que os dispositivos (neste caso lâmpadas) estão ligados, horas do dia em que são acionados cenários e horas do dia em que são acionadas tarefas. Esta informação é preenchida através dos dados registados na tabela "estatísticas" da base de dados do

middleware, que é posteriormente trabalhada, efetuando os devidos cálculos, para cada dos diferentes componentes e gera os respectivos gráficos (Figura 13).

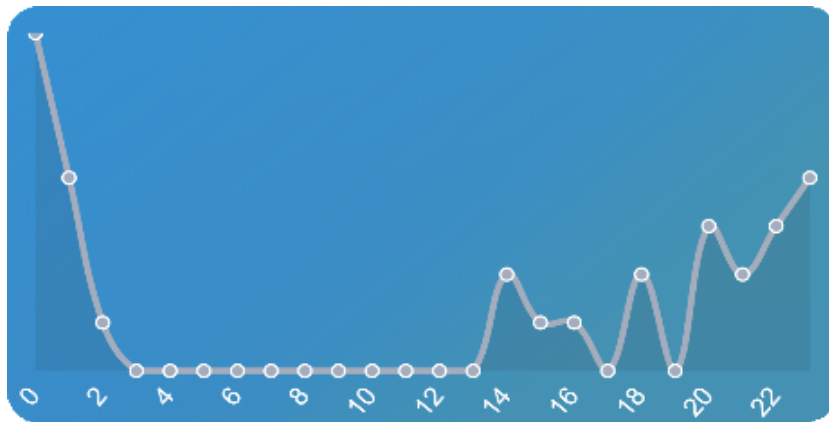


Figura 13: Exemplo de um gráfico de consumo de uma lâmpada durante o dia

#### 4.3.6 Armazenamento local de informação

A maior parte da informação que passa pela aplicação é persistida na base de dados previamente desenvolvida e associada ao middleware. Para chegar a essa informação é sempre necessário consumir serviços, quer sejam eles GET, POST, PUT ou DELETE. Contudo, nem sempre a melhor prática é consumir serviços e consumir recursos externos para trocar informação que poderia estar armazenada nos próprios dispositivos.

Ionic Storage é um plugin para a framework híbrida na qual se desenvolveu a solução e é uma alternativa para o armazenamento de informação nos dispositivos. Neste contexto, este recurso utiliza *SQLite* para guardar a informação, que representa uma base de dados estável e que ao contrário de outras bases de dados, não apaga informação quando os sistemas operativos se encontram com pouco espaço de armazenamento.

A sua utilização é efetuada através de pares "key/value" ou objetos JSON, o que tornou intuitivo e simples a sua integração para armazenar informação. No projeto, a principal vantagem na sua utilização é no processo de autenticação, dado que após o primeiro login a autenticação é guardada em memória, durante um período de tempo, podendo ser eliminado através do processo de logout, e deste modo o utilizador não necessita de efetuar o seu login constantemente. Na listagem 4.12 é representado o funcionamento deste processo, onde no login de um utilizador é armazenado localmente a autenticação com a key "HOMEWATCH\_USER" (linha 4) e durante o logout esse par *key/value* é removido do armazenamento local (linha 8):

```
1 async ionViewWillEnter() {
2     const response = await this.homewatchApi.getApi().users.login(form.value);
3     this.homewatchApi.setAuth(response.data.jwt);
4     this.storage.set("HOMEWATCH_USER", response.data);
```

```

5 }
6
7 async logout () {
8     await this.storage.remove ("HOMEWATCH_USER");
9 }

```

Listagem 4.11: Armazenamento local dos dados de Login

Este recurso foi também utilizado para armazenar a informação de outras propriedades referentes ao utilizador: o seu tema preferencial para a aplicação e o seu idioma. A decisão de guardar esta informação localmente foi tomada com o objetivo de evitar sobrecarga de invocações dos serviços para obter esta informação também armazenada na base de dados, até porque são atributos que são necessários em todas as interfaces gráficas, como será demonstrado na seguinte secção.

#### 4.3.7 Assistência Multi-Idioma

Um dos requisitos previamente definidos consistia na possibilidade de o utilizador poder selecionar o seu idioma preferencial. Neste sentido, foi implementada a funcionalidade de multi-idioma, sendo que os idiomas permitidos são o português, inglês, francês e espanhol.

De forma a satisfazer esta necessidade, foi fundamental utilizar uma biblioteca designada *translate*, que permite que sejam utilizados mecanismos de tradução de diferentes serviços à escolha e traduzir texto com base na linguagem selecionada. Esta API é de utilização gratuita, porventura apresenta um limite máximo de caracteres traduzidos diariamente de 1.000.000 e de 10.000.000 por mês, o que, por ser este um projeto académico e não comercial, são limites completamente compatíveis.

Para utilização desta biblioteca foi necessário num primeiro momento configurar o serviço que se irá recorrer para a tradução, efetuar o devido registo e posteriormente gerar a *API key*. Depois da chave gerada tornou-se possível efetuar invocações do serviço de forma a efetuar tradução do texto. No seguinte exemplo (Listagem 4.12) demonstra-se a tradução de "Homes" para francês utilizando o serviço. É criado um objeto com os parâmetros (linha 8) que posteriormente são enviados na invocação ao serviço (linha 15). A tradução é depois guardada no armazenamento local (linha 1):

```

1 Translate.TranslateText ("Homes", language).then (data => {storage.set ("homesLabel",
2     data)});
3
4 static async TranslateText (input, language) {
5     if (language == "en") {
6         return input;
7     }
8     let params = {

```

```

9      from: 'fr',
10     to: language,
11     engine: 'yandex',
12     key: {{APIkey}}
13   });
14
15   return await translate(input, params);
16 }

```

Listagem 4.12: Tradução de texto

Todas as *labels* e outras componentes textuais presentes na aplicação necessitam de serem dinamicamente traduzidas. Deste modo, e como representado no excerto de código da listagem 4.12, foi necessário implementar um método genérico *TranslanteText()* com o intuito traduzir os diferentes elemento previamente configurados em inglês (linguagem default). As labels são carregadas de forma massiva, e armazenadas localmente no dispositivo, em dois diferentes momentos: durante o processo de login do utilizador, depois de ser identificada o seu idioma preferencial e também no momento em que é alterado o seu idioma, nas *Preferências*. Todos os outros componentes de texto são traduzidos individualmente quando necessário.

#### 4.3.8 Notificações Push

Como já vimos, as notificações push possibilitam o envio de alertas e avisos por parte de uma aplicação diretamente para o dispositivo dos utilizadores. Podem ser informação de atualizações, lembretes, ofertas, notícias ou informação da própria aplicação. No âmbito desta dissertação estas notificações foram utilizadas para o envio de alertas para os dispositivos do utilizador no momento em que tarefas temporais e acionadas são executadas.

Para que estas notificações pudessem ser adicionadas no sistema foi necessário integrar as tecnologias OneSignal e Firebase anteriormente descritas e adicionar lógica para que o envio de notificações fosse realizado de forma totalmente automática. Numa primeira instância é necessário registar os dispositivos do utilizador para que o sistema perceba quais os dispositivos para os quais deverá notificar. A lógica para a adição do dispositivo foi implementada no momento em que é realizado o login dos utilizadores, demonstrado na listagem 4.13, onde se destaca o envio do ID da aplicação e do ID do remetente (linha 2) e o envio do utilizador sobre o qual serão enviadas as notificações(linha 6):

```

1  if (isCordovaAvailable()) {
2    this.oneSignal.startInit(oneSignalAppId, sender_id);
3    this.oneSignal.inFocusDisplaying(this.oneSignal.OSInFocusDisplayOption.
  Notification);
4    this.oneSignal.handleNotificationReceived().subscribe(data => this.
  onPushReceived(data.payload));

```

```

5     this.oneSignal.handleNotificationOpened().subscribe(data => this.onPushOpened(
      data.notification.payload));
6     this.oneSignal.sendTag("user", user.id);
7     this.oneSignal.endInit();
8 }

```

Listagem 4.13: Registo dos dispositivos dos utilizadores

Neste momento o dispositivo é registado no OneSignal, caso ainda não o tenha sido, e associado ao respetivo identificador do utilizador.

As notificações são enviadas para os dispositivos no momento em que as tarefas são executadas. Como tal foi necessário adaptar o código fonte no middleware já desenvolvido para que este processo fosse possível de ser executado. Primeiramente foi necessário adicionar ao projeto a gemfile *onesignal-ruby*, que possibilita o acesso das aplicações desenvolvidas em Ruby à API do OneSignal. Na listagem 4.14 está representado um exemplo de código que representa o envio de notificações para os dispositivos, onde inicialmente são obtidas as variáveis de ambiente armazenadas no servidor (linha 1-linha 3), para posteriormente criar um objeto do tipo OneSignal e aplicar as configurações (linha 5/linha 6). Depois de preenchidos os parâmetros a enviar na notificação (linha 8), a notificação é criada (linha 17) e será enviada através do Firebase:

```

1  api_key=ENV['ONESIGNAL_API_KEY']
2  user_auth_key=ENV['ONESIGNAL_USER_AUTH_KEY']
3  app_id =ENV['APP_ID']
4
5  OneSignal::OneSignal.api_key = api_key
6  OneSignal::OneSignal.user_auth_key = user_auth_key
7
8  params =
9    {
10     "app_id" => app_id,
11     "contents" => {"en" => "Timed Task has been executed."},
12     "filters" => [
13       {"field": "tag", "key": "user", "relation": "=", "value": user.id},
14     ]
15   }
16
17  response = OneSignal::Setting.create(params: params)
18  setting_id = JSON.parse(response.body) ["id"]

```

Listagem 4.14: Invocação da API para envio de notificação

#### 4.3.9 Segurança

Como em qualquer projeto que faça uso de dispositivos IoT, a segurança é um tópico fundamental no desenvolvimento das soluções, e desta forma, desde a fase inicial do processo de desenvolvimento da aplicação, foi pensada uma estratégia de forma a assegurar este tema na aplicação. É importante que a segurança seja garantida não só para os utilizadores, mas também para os programadores e empresas.

Todas as aplicações possuem algum tipo de informação mais crítica e sensível sobre as quais se deverão garantir soluções de forma a evitar que o seu conteúdo seja atacado por terceiros. Nesta solução, a informação mais crítica relativa aos utilizadores são os seus dados de autenticação. Tendo em conta que é um requisito manter a autenticação do utilizador durante um período limitado, é necessário persistir essa informação do lado do cliente. Guardar esta informação na cloud traria alguns problemas de segurança, e como tal a solução passou por utilizar a ferramenta de Ionic Storage, já descrita anteriormente, que permite persistir toda a informação no dispositivo, sendo que esta possui mecanismos próprios de encriptação, garantido a segurança dos dados.

Mas não só a informação crítica provém de dados do utilizador. O próprio sistema necessita de ocultar dados que podem ser facilmente visíveis no código, até por ser esta uma dissertação académica e como tal o código-fonte estará visível num repositório público.

Ao longo destas últimas secções foram mencionados serviços externos que são consumidos pela aplicação nas suas diversas funcionalidades. Por norma, para que sejam consumidos esses serviços, são necessários dados de autenticação por parte do sistema aos serviços, nomeadamente as API keys. Para que fosse possível satisfazer esta necessidade, foi necessário acrescentar lógica no middleware, de modo a que este forneça a informação necessária à aplicação. Assim, sempre que é realizado um pedido para utilização de um serviço, numa primeira fase é pedida a chave da API ao backend da arquitetura, esta devolve a chave codificada em base64, e decodificada do lado da aplicação. Como demonstrado abaixo (listagem 4.15), o método `getKey()` invoca a API do middleware (linha 4) e este retorna a chave em base64 que posteriormente é convertida para decimal (linha 5) :

```
1  async getKey(id)
2  {
3    let homewatchApiService = new HomewatchApiService();
4    const response = await homewatchApiService.getApi().keys.getKey(id);
5    return atob(response.data);
6  }
```

Listagem 4.15: Obtenção de API key através do middleware

Refira-se que este processo não é o mais eficaz para garantir total segurança das chaves da API devido a possíveis ataques de monitorização de tráfego de internet em tempo real, que são ataques onde os dados podem ser interceptados por terceiros e decodificados. O ideal seria a aplicação de um algoritmo de encriptação e desencriptação dos dados para que a tradução pudesse ser realizada tanto pelo middleware como pela aplicação móvel.



Contudo, por não ser o foco da dissertação, considerou-se que o método desenvolvido é suficiente, sendo que a encriptação dos dados ficará para trabalho futuro.

#### 4.4 TESTES

Depois de todas as funcionalidades da aplicação serem implementadas, é necessário realizar uma fase de testes de modo a garantir que o resultado final corresponde ao que foi inicialmente previsto na fase de requisitos e do desenho da solução.

Uma forma de testar a validade de todo o código desenvolvido é através da conceção de testes unitários, que isolam pedaços do código e testam os diferentes componentes, métodos e atributos de forma a encontrar possíveis falhas no projeto. Os testes unitários transmitem uma maior segurança aos programadores, pois garantem que ao longo do desenvolvimento, caso seja colocado código incorreto por exemplo numa função, será alertado através destes testes.

Nesta aplicação, os testes unitários foram realizados recorrendo à framework de testes em JavaScript designada Jasmine [Hahn \(2013\)](#). A grande vantagem da utilização desta framework é a possibilidade que esta possui de isolar componentes, permitindo assim testar partes do código em específico. Esta característica é conseguida graças à utilização de *spies* que permitem simular outros componentes que não são desejados testar no momento, tornando desta forma os testes de métodos de um componente independentes de todo o resto do projeto. De seguida, na listagem 4.16, é apresentado um exemplo de um teste unitário a um dos métodos das funcionalidades associadas às casas, onde são inicialmente criados dois objetos *scenarios* e *scenarioThings* (linha 1/linha 9) que irão corresponder a respostas fictícias de invocações à API (linha 18/linha 19). Posteriormente é invocado o método sobre o qual se pretende efetuar os testes (linha 21) e é verificado o valor da variável de classe *loading* (linha 23):

```
1 let scenarios = {
2   data: [{
3     id: 2,
4     home: 1,
5     icon: "2",
6     (...)
7   }]
8 };
9 let scenarioThings = {
10  data: [{
11    id: 3,
12    status: {on: true},
13    (...)
14  }]
15 };
16
```

```

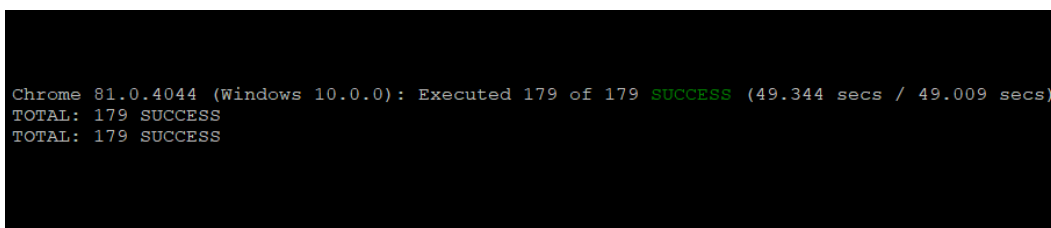
17 it('Load page', fakeAsync(() => {
18     spyOn(component, 'listScenariosAPI').and.returnValue(Promise.resolve(scenarios))
19     ;
20     spyOn(component, 'listScenarioThingsAPI').and.returnValue(Promise.resolve(
21     scenarioThings));
22     component.ionViewWillEnter();
23     tick(3000);
24     expect(component.loading).toBe(false);
25 }));

```

Listagem 4.16: Exemplo de um teste unitário

Juntamente com esta framework é utilizada a biblioteca Karma que fornece as ferramentas necessárias para configurar um ambiente de testes, permitindo preparar todo o processo de execução dos testes unitários. Nestas configurações foi adicionada uma biblioteca designada *coverage-istanbul-reporter* que gera tabelas com a cobertura dos testes sobre o código fonte [Jína \(2013\)](#).

Depois do ambiente de desenvolvimento ter sido configurado foram desenhados e implementados 179 testes unitários para este projeto (Figura 14).



```

Chrome 81.0.4044 (Windows 10.0.0): Executed 179 of 179 SUCCESS (49.344 secs / 49.009 secs)
TOTAL: 179 SUCCESS
TOTAL: 179 SUCCESS

```

Figura 14: Testes Unitários executados com sucesso

A meta estabelecida para os testes deste projeto foi garantir pelo menos 80% de todo o código, sendo que não foi possível garantir uma maior percentagem por parte de algumas funções serem apenas invocações ao middleware e outros serviços, que não deve ser testado neste contexto. Na Figura 15 é demonstrada a cobertura de parte dos testes unitários.

| File                                   | Statements | Branches | Functions | Lines |
|--|------------|----------|-----------|-------|
| src/pages/things/devices/lock          | 100%       | 28/28    | 83.33%    | 15/18 |
| src/pages/things/devices/motion_sensor | 96.15%     | 25/26    | 87.5%     | 14/16 |
| src/pages/scenarios/show               | 95.83%     | 46/48    | 60%       | 3/5   |
| src/pages/tasks/triggered/list         | 95%        | 38/40    | 57.14%    | 4/7   |
| src/pages/things/devices/weather       | 95%        | 38/40    | 90%       | 9/10  |
| src/pages/users/homepage               | 94.18%     | 178/189  | 61.11%    | 11/18 |
| src/pages/statistics/thing_stats       | 94.06%     | 206/219  | 60.29%    | 41/68 |
| src/pages/rooms/new                    | 94.03%     | 63/67    | 55.56%    | 5/9   |
| src/pages/tasks/list                   | 93.91%     | 185/197  | 74.51%    | 38/51 |

Figura 15: Cobertura dos testes unitários

## 4.5 DEPLOYMENT

Esta secção representa parte da fase final do processo de desenvolvimento da aplicação híbrida. Após todas as funcionalidades terem sido desenvolvidas e os testes unitários executados com sucesso, é necessário preparar a aplicação para que possa ser distribuída aos utilizadores.

Na fase de deployment é importante configurar o ambiente de produção onde a aplicação será alojada e identificar as dependências de serviços externos, garantindo que o produto será disponibilizado nas melhores condições.

Aqui será representado todo o ambiente produtivo configurado, desde a alocação do código fonte da aplicação em repositórios públicos, a configuração do servidor onde estará a correr o backend da aplicação e a construção do pacote de instalação da aplicação para dispositivos móveis.

### 4.5.1 Repositório público

Todo o processo de desenvolvimento da solução foi acompanhado por um sistema de controlo de versões, nomeadamente o Git, juntamente com o GitHub, o serviço de hosting do Git. A utilização desta ferramenta para além de permitir ter um controlo das diferentes versões do código implementado, oferece a possibilidade de tornar todo o repositório de acesso público e de fácil acesso.

Os diferentes componentes da arquitetura foram publicados nos seguintes repositórios:

- **Dispositivos** - <https://github.com/LeonelGoncalves12/homewatch-devices>
- **Hub** - <https://github.com/LeonelGoncalves12/homewatch-hub>
- **API** - <https://github.com/LeonelGoncalves12/homewatch-api>
- **Wrapper** - <https://github.com/LeonelGoncalves12/homewatch-wrapper>

- **Aplicação cliente** - <https://github.com/LeonelGoncalves12/homewatch-app>

É de realçar que a API, o wrapper, o HUB e os simuladores de dispositivos são componentes inicialmente desenvolvidos na dissertação anterior, contudo, como no âmbito desta dissertação alguns desses elementos sofreram algumas modificações, descritas anteriormente neste documento, foi necessário também colocar todo este código em novos repositórios.

#### 4.5.2 Heroku

Heroku é uma plataforma cloud-based que fornece serviços de hosting, permitindo desta forma hospedar aplicações em variadas linguagens. Esta plataforma possui os seus próprios sistemas operativos e espaço de armazenamento, possibilitando deste modo que seja armazenado não só o código da aplicação, como também bases de dados ou bibliotecas. No âmbito desta dissertação, esta plataforma irá ser utilizada para ser realizado parte do backend da arquitetura.

Para além de disponibilizar serviços para manter a aplicação online, esta plataforma permite a configuração das variáveis necessárias no ambiente produtivo, desde definições, como por exemplo a autorização para escrita de logs, ao armazenamento de diferentes chaves, nomeadamente as de serviços externos.

Neste processo de deploy a integração com o GitHub é crucial devido à possibilidade que a plataforma oferece em selecionar um repositório existente no GitHub e realizar deploys de forma autónoma e com o mínimo esforço dos desenvolvedores. Esta característica torna o processo de manutenção das aplicações mais simples e eficaz graças à realização de builds automáticos a cada commit realizado no repositório.

#### 4.5.3 APK

Depois do servidor configurado foi necessário publicar a aplicação desenvolvida para que pudesse ser utilizada num ambiente produtivo, pelos utilizadores. Desta forma, foi necessário produzir um pacote pronto a ser executado pelos dispositivos e a ser instalado nos mesmos.

No âmbito desta dissertação foi apenas realizado o deploy para Android, contudo os procedimentos para deploys noutros sistemas operativos são semelhantes entre si. O deploy para Android e Windows pode ser executado através de uma máquina Windows, contudo para iOS é imperativo que este processo seja realizado através do sistema operativo MacOS, pois é necessário criar uma conta iOS para que este processo seja exequível. O deploy das aplicações móveis é um processo bastante simplificado graças à Framework que foi utilizada nesta dissertação, necessitando apenas de executar os seguintes comandos:

- **Android** - `ionic cordova build --release android`
- **IoS** - `ionic cordova build --release ios`

- **Windows** - *ionic cordova build --release windows*

Depois destes scripts serem executado é gerado o pacote de instalação para cada um dos sistemas operativos e fica disponível para a respetiva instalação.

---

## RESULTADO FINAL

---

Este capítulo irá servir para apresentar a aplicação híbrida que foi desenvolvida no contexto desta dissertação, demonstrando as diversas funcionalidades que foram inicialmente idealizadas e posteriormente implementadas. Também aqui será detalhado o ambiente que foi utilizado nesta demonstração, desde a definição do Hub, passando pelos simuladores de dispositivos IoT até ao smartphone utilizado.

### 5.1 AMBIENTE DE EXECUÇÃO

Para que fosse possível realizar a demonstração da aplicação a ser executada, o mais próximo possível de um ambiente real onde existe uma box pré-configurada com o software do Hub, foi necessário simular o comportamento de uma casa inteligente instalando o Hub num *Raspberry Pi*, que corre no sistema operativo *Raspbian*. Este Hub estará a correr em todo o momento e preparado para receber e enviar informação para a parte lógica da aplicação.

Nesta dissertação foram utilizados seis simuladores de dispositivos de Internet of Things, em que cinco deles correspondem a simuladores desenvolvidos na dissertação anterior e o último corresponde a um simulador de uma lâmpada Philips Hue. Sendo que correspondem a simuladores, eles estarão diretamente instalados no mesmo sistema onde está instalado o Hub, pronto a serem pesquisados e instalados. A aplicação híbrida, apesar de estar preparada para ser executada em qualquer sistema operativo, será testada apenas em Android, utilizando o dispositivo Huawei P20 Pro.

### 5.2 DEMONSTRAÇÃO

Esta secção tem como objetivo efetuar a demonstração da aplicação desenvolvida e de visualizar as suas funcionalidades colocadas em prática. Estas funcionalidades referem-se ao login e registo de um utilizador, a configuração da sua casa, divisões e dispositivos, a criação de cenários de utilização e de tarefas, e também à definição das suas preferências e à visualização das suas estatísticas e meteorologia. Todas estas funcionalidades são representadas através de ecrãs móveis relativos ao sistema operativo Android, de modo a não estender demasiado

esta secção com a apresentação para outros sistemas, sendo que mais à frente neste capítulo será feita uma comparação entre os ecrãs nos outros SO's.

Para começar esta demonstração, é necessário primeiramente efetuar o registo do utilizador (Figura 16).

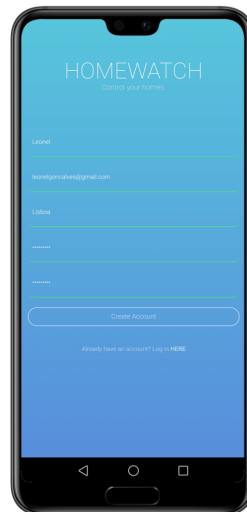


Figura 16: Registo do Utilizador

Depois do registo efetuado, o utilizador fica registado no sistema e está credenciado para efetuar o login na aplicação (Figura 17).

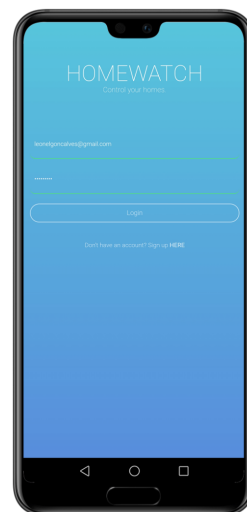


Figura 17: Login do Utilizador

O login redireciona o utilizador para a homepage, onde este tem uma visão geral do sistema e das páginas que poderá aceder, desde a gestão das suas habitações, as estatísticas, e as definições e preferências. É também na homepage onde são apresentadas as condições meteorológicas para a cidade definida pelo utilizador (Figura 18).



Figura 18: Homepage

Neste momento o utilizador não possui nenhuma casa registada. Avançou-se para o registo de uma casa, onde são preenchidas as suas informações e o endereço do *tunnel* pelo qual se irá comunicar com o Hub (Figura 19).

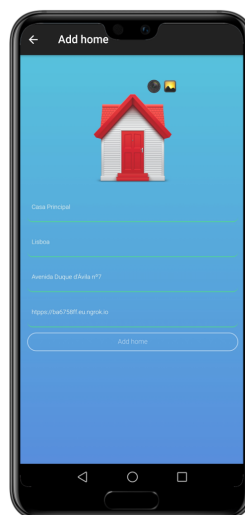


Figura 19: Instalar uma casa



Depois de instalada a sua casa, o utilizador tem acesso à visão geral da sua casa, onde pode visualizar todas as suas divisões, dispositivos, cenários e tarefas configuradas (Figura 20).

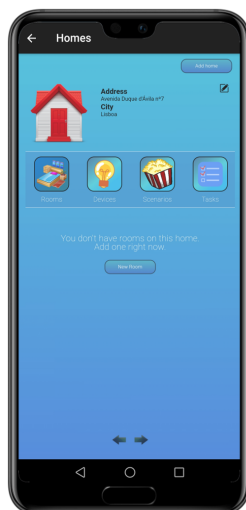


Figura 20: Visão geral das casas do utilizador

Antes de iniciar a configuração de dispositivos, é necessário primeiramente adicionar as respetivas divisões. Estas estão já pré-configuradas no sistema juntamente com os respetivos ícones a apresentar. Depois das divisões terem sido configuradas, as mesmas podem ser acedidas através do ecrã de visão geral da casa (Figura 21).



Figura 21: Divisões de uma casa

As diferentes divisões estão registadas e agora é necessário registar os diferentes dispositivos nas mesmas (Figura 22). Existem 5 tipos pré-configurados no âmbito desta dissertação: Lâmpadas, Fechaduras, Termostatos, Sensores de Movimento e Meteorologia (com sensores de temperatura, vento, chuva e céu nublado). Para esta demonstração foi instalado cada um deles. Para isso é necessário selecionar o tipo de dispositivo a adicionar e pesquisar pelo dispositivo. Caso o mesmo seja encontrado pelo Hub, é adicionado no sistema.

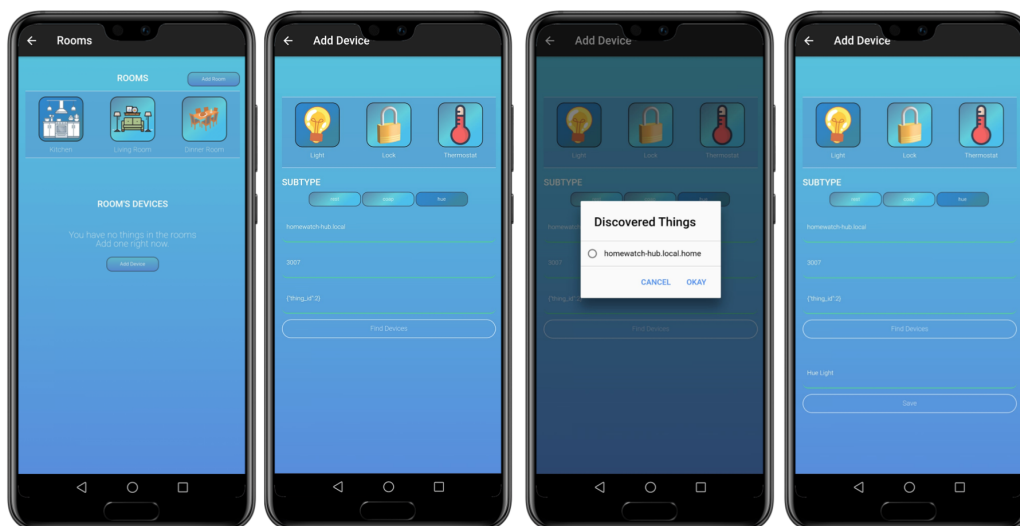


Figura 22: Adição de novos dispositivos

O estado dos dispositivos pode ser alterado através de tarefas acionadas, tarefas cronometradas ou simplesmente de forma individual. Para além da visualização do estado dos dispositivos através da aplicação, é possível verificar a alteração de estado dos dispositivos na consola do middleware (Figura 23).

```
[2020-10-18T21:45:34.889Z] RECEIVED GET REQUEST: {}  
[2020-10-18T21:46:39.623Z] RECEIVED GET REQUEST: {}  
[2020-10-18T21:46:44.577Z] RECEIVED PUT REQUEST: {"power":true}  
[2020-10-18T21:46:52.612Z] RECEIVED PUT REQUEST: {"power":false}  
[2020-10-18T21:46:58.662Z] RECEIVED PUT REQUEST: {"power":true}  
[2020-10-18T21:47:00.422Z] RECEIVED GET REQUEST: {}  
[2020-10-18T21:47:05.584Z] RECEIVED GET REQUEST: {}
```

Figura 23: Consola do middleware após mudar o estado de um dispositivo

Através da consola do middleware é também visível a receção de pedidos através do tunel para que a aplicação tenha acesso ao estado dos dispositivos (Figura 24).

```

HTTP Requests
-----
GET /devices/lights           200 OK
GET /devices/locks           200 OK
GET /devices/lights           200 OK

```

Figura 24: Receção de pedidos no túnel do middleware

Para cada divisão são apresentados os dispositivos configurados e os respetivos estados dos mesmos: a cor verde representa um dispositivo ligado, a cor vermelha representa um dispositivo desligado, e a cor cinza representa um dispositivo que não apresenta este tipo de sensor (por exemplo Sensor de Movimento). A cor cinzenta poderá também representar um dispositivo cujo estado não foi possível obter no momento por alguma indisponibilidade (Figura 25).

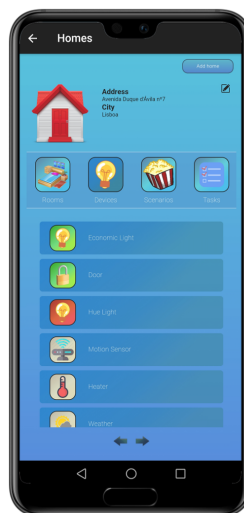


Figura 25: Visão geral dos dispositivos

Com os dispositivos já configurados na nossa casa, podemos registar cenários (Figura 26). Os cenários representam conjuntos de dispositivos e os estados que os mesmos devem assumir de uma só vez. Neste processo é necessário selecionar um ícone para representar o cenário e dar um nome ao cenário. Aqui será criado um cenário "Turn on all Lights", com o objetivo de ligar todas as lâmpadas de uma só vez.



Figura 26: Criação de um cenário

Após o cenário criado, é necessário adicionar os dispositivos e os respectivos estados. Neste cenário da Figura 27 foi adicionada a lâmpada da cozinha com o estado "ligado" e a lâmpada Hue da Sala de Estar com o estado "ligado"

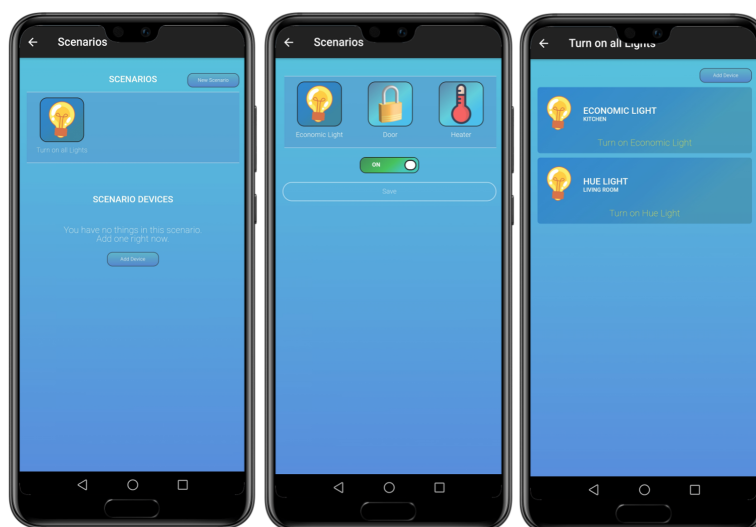


Figura 27: Adição de dispositivos aos cenários

Para além dos cenários é possível configurar também tarefas, e aqui há dois tipos de tarefas: as temporais e as acionadas por dispositivos. Nesta demonstração será criada uma tarefa de cada uma destas tipologias.

Uma tarefa temporal pode ser aplicada a dispositivos e a cenários, sendo que quando utilizados dispositivos, devem ser declarados os estados que os mesmos deverão assumir. Seguidamente deverá ser preenchido o horário em que esta tarefa deverá ser executada. Nesta demonstração, na Figura 28, foi criada a tarefa de regular a temperatura do termostato a 22°C, todos os dias à 1h.

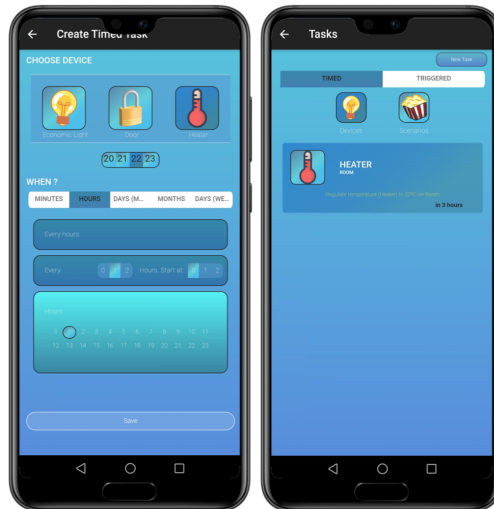


Figura 28: Criação de uma Tarefa Temporal

Após a tarefa ser executada, uma notificação é lançada no dispositivo do utilizador (Figura 29).

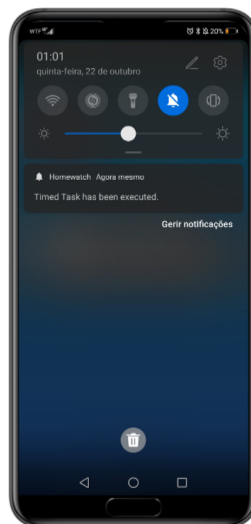


Figura 29: Notificação de uma tarefa cronometrada

Através da consola do middleware, é possível verificar que o dispositivo alterou o seu estado no horário para o qual a tarefa foi definida (Figura 30).

```
[2020-10-22T00:00:07.694Z] RECEIVED GET REQUEST: {}
[2020-10-22T00:00:13.311Z] RECEIVED PUT REQUEST: {"target_temperature":22}
```

Figura 30: Consola do middleware após tarefa ser executada

De seguida foi criada uma tarefa que seja acionada por um dispositivo. Também aqui poderão ser definidas tarefas para dispositivos e para cenários e da mesma forma deverá ser definido um estado a aplicar no caso dos dispositivos (Figura 31). Para que esta tarefa seja executada, deverá ser também estabelecido o estado do dispositivo que acionará a tarefa. Aqui foi definida a tarefa de fechar a fechadura da porta da Sala quando a temperatura da Sala for de 25°C.

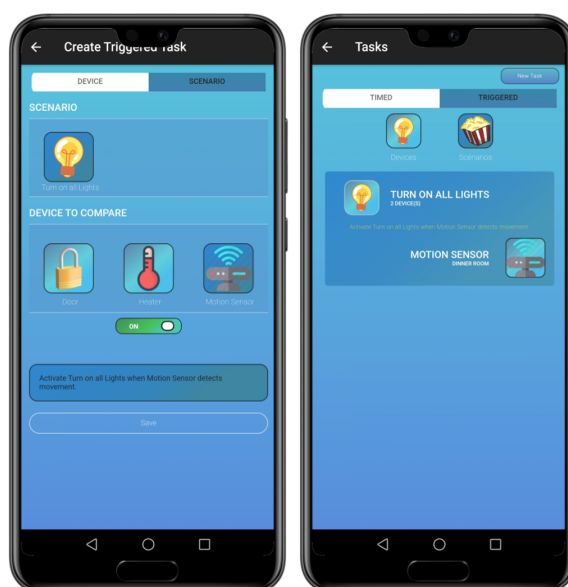


Figura 31: Criação de uma Tarefa Acionada por Dispositivos

Quando uma tarefa é executada, independentemente da sua tipologia, o utilizador é notificado através de uma notificação push no seu dispositivo.

As funcionalidades acima representadas correspondem aos requisitos definidos como prioritários. Para não estender demasiado esta demonstração, não serão aqui apresentadas as funcionalidades relativas à edição e remoção dos diversos componentes. Existem, contudo, outras funcionalidades que a aplicação fornece, cujos requisitos apresentavam uma prioridade inferior.

De forma a garantir uma melhor gestão das suas casas inteligentes, o utilizador pode visualizar as suas estatísticas de utilização de dispositivos, cenários e tarefas, através de gráficos que revelam as horas que estes mais são utilizados e de outros dados, como os consumos médios para alguns dispositivos (Figura 32)



Figura 32: Estatísticas do Utilizador

O utilizador poderá também alterar a sua informação pessoal, ativar e desativar notificações e definir as suas preferências, nomeadamente o idioma e o tema da aplicação (Figura 33)



Figura 33: Definições e Preferências

---

## ANÁLISE DOS RESULTADOS

---

Os resultados obtidos no desenvolvimento desta aplicação foram bastante satisfatórios e coincidem com o que foi inicialmente idealizado na fase de análise e concepção do projeto. Neste capítulo serão apresentados os pontos principais identificados na utilização desta ferramenta híbrida numa solução para Smart Homes.

### 6.1 IMPLEMENTAÇÃO DAS FUNCIONALIDADES PREVISTAS

A implementação desta aplicação foi antecedida de um processo de estudo de tecnologias similares nesta área tecnológica e uma posterior definição de requisitos funcionais e não funcionais, com a respetiva prioritização de desenvolvimento. Este processo facilitou a implementação de todo o sistema, pois permitiu num primeiro momento isolar os componentes base para gestão de uma casa inteligente, desde a adição de casas, dispositivos cenários e tarefas, e posteriormente partir para o desenvolvimento de outras funcionalidades, como por exemplo as estatísticas, a informação sobre a meteorologia ou as preferências do utilizador.

O objetivo do extenso número de funcionalidades previstas tinha como objetivo assemelhar o mais possível a aplicação a um contexto real, para que na fase de testes da mesma fosse possível tirar conclusões mais fidedignas sobre a utilização desta tipologia de aplicações em contexto de casas inteligentes. Esta decisão foi proveitosa nesse sentido, contudo tornou o tempo de desenvolvimento da aplicação relativamente maior.

As funcionalidades foram desenvolvidas ao longo de diversas linhas de código, e como tal, para que se garantisse que o código implementado não possui falhas, o sistema foi revestido através de testes unitários que foram na sua totalidade executados com sucesso.

### 6.2 INTEGRAÇÃO COM O BACKEND

Como já referido anteriormente, a componente lógica da arquitetura encontra-se hospedado num servidor Cloud, e como tal, toda a comunicação com o mesmo necessita de ser realizada através de uma API. Graças ao *wrapper* concebido na dissertação anterior que converte as chamadas à API utilizando HTTP para funções em JavaScript,



a integração com a framework híbrida foi um processo bastante simplificado. Caso esse *wrapper* não estivesse já desenvolvido seria necessário construir um semelhante nesta dissertação, contudo, após analisar todo o código do mesmo, constatou-se que em JavaScript a conversão de invocações HTTP para esta linguagem é bastante mais simples de se desenvolver do que em outras linguagens para desenvolvimento nativo, como Java e C#.

Uma vantagem que este tipo de arquitetura possui, onde foi separado o backend do frontend, é a independência que o isolamento das aplicações é capaz de fornecer, contudo com algumas limitações do lado do frontend. Caso exista alguma indisponibilidade do servidor, a aplicação continua executável, contudo algumas funcionalidades ficam indisponíveis devido ao facto das invocações dos Web Services não retornarem qualquer tipo de informação.

### 6.3 EXPERIÊNCIA DE UTILIZAÇÃO

Uma das características que se procura através da utilização de Smart Homes é o conforto que se deseja obter através da mesma. Como tal, sempre foi um objetivo garantir aspectos que pudessem proporcionar ao utilizador uma experiência de utilização favorável. Algumas das características da aplicação implementadas com o objetivo de garantir qualidade na utilização da aplicação foram:

- Utilização de ícones simples e apelativos;
- Alternância entre interfaces de um modo fluido;
- Possibilidade de seleção de tema e idioma preferencial;
- Notificações de Alerta;
- Possibilidade de utilização da aplicação independente da localização.

Um factor que melhora a experiência de utilização de qualquer aplicação é a capacidade que o utilizador possui para executar uma função com o mínimo esforço possível. Tendo isto em conta, foram implementadas pequenas funcionalidades ao longo da aplicação, desde a capacidade de fornecer uma visão geral da casa, atalhos para algumas das páginas, troca entre ecrãs através de *swipe*, entre outras. Para trabalho futuro introduz-se neste ponto a integração da framework híbrida utilizada com outros recursos nativos de forma a desenvolver *widgets* para dispositivos, possibilitando a execução de tarefas sem ter sequer de inicializar a aplicação.

Para além do conforto e redução de esforço, o utilizador procura também através da utilização de casas inteligentes uma redução de custos. As estatísticas de utilização foram desenvolvidas com este objetivo, contudo aqui foram encontradas algumas adversidades durante o desenvolvimento. Dado que este ponto não era um requisito prioritário, os detalhes de consumo foram desenvolvidos apenas para dispositivos de iluminação, sendo que para estes foram definidos valores base quer para consumo por hora, quer para o preço por consumo. Estes valores podem não corresponder à realidade, apesar dos valores assumidos corresponderem a valores médios para uma lâmpada.

#### 6.4 DESEMPENHO DOS RECURSOS NATIVOS

Como foi referido na fase de demonstração, a aplicação desenvolvida foi testada apenas num dispositivo físico que corria no sistema operativo Android, e como tal, a maioria dos recursos nativos utilizados nesta aplicação foram apenas testados tendo em consideração este sistema. Contudo, aqui os resultados são bastante satisfatórios, não tendo sido detetado nenhum problema durante a utilização das APIs de Geolocalização, Câmara/Galeria e Vibração. A lógica das funções relativas a estas implementações não variou consoante o sistema operativo e deste modo espera-se que funcionem de igual forma noutros SO's.

Outro recurso nativo utilizado nesta dissertação foram as *Notificações Push* no momento em que as tarefas são executadas. Estas notificações são recebidas entre 5-10 segundos após as notificações terem sido notificadas, não tendo sido detetada em nenhum dos testes realizados alguma indisponibilidade que ditou o não envio da notificação.

#### 6.5 UNIFORMIDADE DAS INTERFACES ENTRE MÚLTIPLAS PLATAFORMAS

Cada sistema operativo, na utilização de qualquer aplicação móvel, possui as suas próprias características no que diz respeito aos respectivos elementos de User Interfaces, como por exemplo as barras de navegação, botões, menus ou separadores, existindo por vezes ligeiras diferenças na forma como estes elementos se comportam. Dado que esta dissertação trata de aplicações híbridas e dessa forma a aplicação deverá ser executada em múltiplos sistemas, foi necessário arranjar um meio termo entre uniformidade de ecrãs da aplicação ao longo dos diferentes sistemas operativos e a usabilidade que os utilizadores destes sistemas operativos estão acostumados durante a utilização de outras aplicações.

Neste ponto, foi assim necessário desenvolver um *style* específico para alguns elementos nas diferentes plataformas, como foi o caso das barras de navegação e separadores (Figura 34).



Figura 34: User Interfaces nas diferentes plataformas

## 6.6 DEPENDÊNCIAS THIRD-PARTY

Ao longo da dissertação foram descritas as diversas dependências externas aplicadas neste projeto, desde Web Services a bibliotecas JavaScript. A utilização destes recursos pode ser analisada da seguinte forma:

- **Web Services** - A utilização destes serviços possibilitou que a aplicação se tornasse mais rica no que toca às funcionalidades que a mesma é capaz de fornecer. A integração destes serviços na arquitetura foi um processo facilitado graças à abundante documentação e apoio da comunidade, consequência da enorme popularidade da linguagem JavaScript. Neste momento, o único entrave na utilização dos Web Services é estarem a ser utilizados planos *free*, o que na maioria dos casos leva a um limitado número de invocações à API que se podem realizar.
- **Bibliotecas** - Através do *Node Package Manager* foram obtidas diversas bibliotecas para que fossem integradas neste projeto, sendo que graças à sua utilização levou a uma grande redução de tempo de desenvolvimento da aplicação.

Apesar desta integração ter visíveis vantagens no projeto, não deixam de ser dependências, o que significa que qualquer problema poderá prejudicar o correto funcionamento da aplicação. Estes problemas, no caso dos serviços podem consistir em indisponibilidades dos mesmos, e no caso das bibliotecas podem expressar-se através

de atualizações que causem incompatibilidades entre si ou com o resto do sistema. Apesar de durante a fase de testes nenhum destes problemas ter sido detetado, é uma realidade que num ambiente produtivo poderá acontecer.

## 6.7 MANUTENÇÃO DO CÓDIGO

Apesar de várias funcionalidades terem sido desenvolvidas, no contexto de casas inteligentes existem diversas melhorias que podem ser ainda implementadas no projeto, e como tal toda a estrutura da aplicação foi pensada para que se garantisse a melhor organização e legibilidade do código possível para eventuais modificações. As páginas estão distribuídas ao longo dos respetivos componentes enquanto que os módulos reutilizáveis entre diversas classes existentes fazem parte dos *helpers* da aplicação. Esta disposição, facilitada através do uso de Ionic Framework, garante que qualquer programador consiga entender as funcionalidades desenvolvidas sem realizar esforços adicionais. Também esta framework, por ser híbrida, torna todo o processo de manutenção do código e de novos desenvolvimentos menos custoso, em consequência de ser necessário alterar código de apenas uma única aplicação.

---

## CONCLUSÕES E TRABALHO FUTURO

---

Neste capítulo pretende-se elaborar as considerações finais desta dissertação, descrevendo também os desafios e dificuldades encontradas ao longo de todo o projeto. Serão ainda abordadas as funcionalidades que podem ser desenvolvidas no futuro, visto que este software ainda se encontra numa fase inicial do seu ciclo de vida e possui vários pontos onde pode ser melhorado após o término desta dissertação.

### 7.1 CONSIDERAÇÕES FINAIS

Finalizada a implementação da aplicação e realizada a análise dos resultados obtidos, é o momento de realizar uma retrospectiva de todo o trabalho elaborado ao longo dos processos de análise e conceção da aplicação e consequentemente retirar as respetivas conclusões sobre o tema desta dissertação.

Esta dissertação teve como principal finalidade analisar o comportamento das aplicações híbridas no contexto de uma solução para casas inteligentes recorrendo à elaboração de uma aplicação nesse sentido. A aplicação implementada deu seguimento a uma dissertação previamente desenvolvida, na qual são fornecidos os serviços para que sejam consumidos por qualquer outra aplicação. A primeira etapa desta dissertação consistiu em explorar esse trabalho elaborado e replicar o ambiente de desenvolvimento documentado, para que se conseguisse avançar com o desenvolvimento da aplicação móvel.

Todo este processo foi iniciado através da realização do estudo dos diversos tópicos que se pretendia abordar ao longo da dissertação, começando por perceber o que é Internet of Things e como este conceito se insere nas Smart Homes. No estudo das Smart Homes, a análise de tecnologias já existentes neste setor foi uma tremenda ajuda para que se identificassem as funcionalidades base que devem estar presentes neste tipo de aplicações. Também durante esta fase foram exploradas as diferentes tipologias de aplicações móveis, identificando não só as suas valências, mas também os seus pontos mais fracos. A investigação realizada nesta fase permitiu adquirir conhecimento sobre o tema desta dissertação e com o tempo aumentou o interesse e a motivação para posterior desenvolvimento da aplicação. Nesta fase ficou evidente que seria necessário adicionar alguma lógica ao código existente no middleware previamente definido, de forma a garantir um maior número de funcionalidades na aplicação.

Concluído o estudo das diversas temáticas, seguiu-se a fase onde se desenhou aquilo que seria a aplicação móvel a desenvolver, começando por se definir uma lista de requisitos e a respetiva prioridade de desenvolvimento, e posteriormente, através de diversos modelos, se definiu a arquitetura da aplicação e como os seus elementos se iriam interligar. Como em qualquer desenvolvimento de software, esta fase foi crucial, pois a existência de falhas naquilo que seria o desenho técnico e funcional da aplicação a desenvolver poderia futuramente levar ao seu fracasso. Concluído o processo de desenvolvimento e olhando para o que foi definido nesta fase, percebe-se que alguns dos requisitos implementados tiveram uma prioridade superior à que realmente deveriam ter tido, como foi o caso da criação de estatísticas para o utilizador e das definições da aplicação e das notificações que deveriam ter ficado como trabalho futuro, dado que a sua implementação levou a que o tempo de desenvolvimento do projeto fosse superior ao inicialmente esperado.

Com toda a arquitetura da solução delineada, chegou a fase mais longa de toda a dissertação: a implementação da aplicação. Esta etapa foi iniciada com a definição de todas as tecnologias que seriam necessárias de modo a implementar as funcionalidades delineadas, desde Frameworks a Web Services a utilizar. Seguindo a prioritização dada aos requisitos, foram primeiramente desenvolvidas as funcionalidades referentes aos requisitos obrigatórios, deixando para o final os que seriam menos prioritários. Esta técnica permitiu seguir uma ordem de desenvolvimento da aplicação, pois desta forma não existiu a necessidade de iniciar o desenvolvimento de várias funcionalidades simultaneamente, como acontece por vezes neste tipo de projetos académicos.

É boa prática de qualquer aplicação garantir que todas as funcionalidades serem testadas, e como tal, a etapa que se seguiu passou pela realização de testes unitários, onde através do isolamento de partes do código, permitiu assegurar o correto funcionamento da aplicação implementada na fase anterior. Neste sentido, e como o objetivo de qualquer teste unitário é apenas testar as funcionalidades e não a arquitetura como um todo, foi necessário simular as respostas das invocações ao backend do projeto, mantendo as respostas simuladas o mais próximo possível a uma invocação real. Os diversos testes realizados permitiram encontrar algumas falhas no sistema, o que realça ainda mais a importância desta fase. Os testes unitários servirão também para garantir que possíveis alterações no código no futuro não tenham qualquer impacto no correto funcionamento da aplicação.

Depois da aplicação implementada e testada seguiu-se a fase de deployment, onde foi realizada a passagem de todo o código desenvolvido para um ambiente produtivo. Aqui foi necessário não só transformar a aplicação realizada no âmbito desta dissertação num pacote pronto a instalar em Android, mas também colocar o servidor online, sendo que este conta com as alterações realizadas durante este projeto. Este último passo foi simplificado por se ter seguido o processo realizado na dissertação anterior, quando este foi disponibilizado na Cloud.

Concluídas todas estas etapas, o sistema estava preparado para ser testado como um todo e realizar a devida análise dos resultados obtidos, de modo a retirar conclusões concretas acerca da utilização de aplicações híbridas no universo das Smart Homes. Nesse sentido, as considerações finais são as seguintes:

- Uma das principais mais valias encontradas neste domínio foi a possibilidade que é oferecida ao utilizador ao poder alterar o dispositivo com que gere a sua habitação ao longo do tempo, garantindo que o utilizador não tenha de se vincular a um único sistema operativo.
- Ainda que as aplicações nativas proporcionem um melhor desempenho, o que esta aplicação híbrida assegurou durante a execução das diversas funcionalidades é extremamente satisfatório, sem ter sido relatado nenhuma falha neste sentido. Conclui-se deste modo que não é necessário apostar na obtenção do máximo desempenho possível, através de aplicações nativas, numa aplicação para este domínio.
- Como foi visto anteriormente, a IoT e as Smart Homes continuam numa evolução exponencial, não só nos seus equipamentos mas também nas aplicações que permitem a sua gestão. É assim previsível que após a entrada da aplicação num ambiente de produção, esta passe por diversas manutenções e evoluções. Estes processos são bastantes simplificados pelo facto de ser necessário efetuar alterações numa única aplicação e também graças à facilidade com que o deployment da mesma é efetuado para os diferentes sistemas operativos.
- Ficou comprovada a facilidade com que as APIs nativas conseguem ser utilizadas, nomeadamente a Câmara, Geolocalização e a Vibração, recursos que apesar de não serem indispensáveis neste contexto, possibilitam a adição de funcionalidades úteis para o utilizador.
- O código fonte da aplicação pode ser na íntegra contruído durante o desenvolvimento da aplicação, contudo existem inúmeras bibliotecas que proporcionam os recursos desejados e que podem ser facilmente integradas em projetos no setor casas inteligentes. JavaScript, a linguagem utilizada no desenvolvimento de aplicações híbridas, é rica no que toca ao número de bibliotecas open-source, sendo que estas são facilmente integradas nos projetos, o que é uma mais valia para o desenvolvimento de aplicações.

## 7.2 TRABALHO FUTURO

Esta secção tem como objetivo abordar alguns aspetos que na sequência desta dissertação podem ser melhorados. Os objetivos inicialmente definidos foram concluídos com sucesso, contudo existem alguns pontos que podem ser otimizados, de forma a sustentar o crescimento da aplicação, uma vez que as tecnologias estão em contante evolução:

- apesar de terem sido contemplados cinco tipos de dispositivos tanto na construção do middleware, como nesta dissertação, existem muitos outros que podem ser incluídos neste sistema, como por exemplo televisões ou câmaras de segurança. No futuro tenciona-se adicionar alguns destes equipamentos no projeto.
- existem propriedades que podem ser adicionadas aos elementos do sistema, como por exemplo as divisões possuírem, tal como numa casa, as respetivas imagens, ou os dispositivos possuírem a identificação da sua

marca de fabricante. A adição dessas propriedades permite assim a implementação de novas funcionalidades.

- o processo de criação de tarefas temporais pode suscitar algumas dúvidas no utilizador relativamente ao seu modo de funcionamento. Deste modo, será alterado visualmente esta página, tornando-a menos confusa.
- desenvolver um processo de encriptação e desencriptação do lado do middleware e da aplicação para melhorar a segurança no momento de troca de chaves de API's.
- para que o utilizador consiga utilizar algumas funcionalidades sem ter a necessidade de abrir a aplicação, pretende-se implementar *widgets* nativos simples, como por exemplo para ligar/desligar dispositivos.
- implementação de pequenas *tooltips* na maioria das páginas para que o utilizador possa tirar possíveis dúvidas durante a utilização do sistema.
- as notificações push foram apenas desenvolvidas para alertarem o utilizador que as tarefas foram executadas. Esta lógica pode ser reutilizada para alertar os utilizadores de outras situações, como por exemplo dispositivos ligados há demasiado tempo.



---

## BIBLIOGRAFIA

---

- K. S. Ahmad, N. Ahmad, H. Tahir, and S. Khan. A fuzzy based moscow method for the prioritization of software requirements. In *2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICT)*, pages 433–437, 2017. doi: 10.1109/ICICT1.2017.8342602.
- Alarm.com. What do homeowners really want from the smart home?, 2016. URL <https://www.alarm.com/blog/smart-home-survey-homeowners-parents>.
- Apple. Apple developer documentation - homekit, 2019a. URL <https://developer.apple.com/documentation/homekit>.
- Apple. ios - home apple, 2019b. URL <https://www.apple.com/ios/home/>.
- Chandra Bhupesh and Bhupesh Joshi. Hybrid mobile application based on ionic framework technologies. 07 2019.
- Andrea Sánchez Blanco. *Development of Hybrid Mobile apps Using Ionic Framework*. PhD dissertation, Mikkeli University of Applied Sciences, 2016.
- Jeannette Chin, Vic Callaghan, and Somaya Allouch. The internet-of-things: Reflections on the past, present and future from a user-centered and smart environment perspective. *Journal of Ambient Intelligence and Smart Environments*, 11(1):45–69, 2019. doi: 10.3233/AIS-180506.
- José Francisco Ferreira Alves de Sousa. *Definição de middleware "cloud-based" para serviço a aplicações de monitorização de espaços físicos*. Masters dissertation, Universidade do Minho, 2017.
- Tam Doan, Reihaneh Safavi-Naini, Shuai Li, Sepideh Avizheh, Muni Venkateswarlu K, and Philip Fong. Towards a resilient smart home. pages 15–21, 08 2018. doi: 10.1145/3229565.3229570.
- Menachem Domb. *Smart Home Systems Based on Internet of Things*. 02 2019. doi: 10.5772/intechopen.84894.
- Bakwa Dunka, Edim Emmanuel, and Yinka Oyerinde. Hybrid mobile application based on ionic framework technologies. *International Journal of Recent Advances in Multidisciplinary Research*, 04:3121–3130, 12 2017.
- Aleksandar Georgiev and Stephan Schlögl. Smart home technology: An exploration of end user perceptions. 02 2018a.
- Aleksandar Georgiev and Stephan Schlögl. Smart home technology: An exploration of end user perceptions. 02 2018b.

- Oliver Gill. *Using React Native for mobile software development*. PhD dissertation, Metropolia University of Applied Sciences, 2018.
- Evan Hahn. *JavaScript testing with Jasmine*. "O'Reilly Media, Inc.", 2013.
- Tom Hargreaves and Charlie Wilson. Who uses smart home technologies? representations of users by the smart home industry. 8, 2013.
- Amjad Hudaib, Raja Masadeh, Mais Haj Qasem, and Abdullah Alzaqebah. Requirements prioritization techniques comparison. *Modern Applied Science*, 12, 01 2018. doi: 10.5539/mas.v12n2p62.
- IEEE. Iot - developer survey, 2016. URL <https://iot.ieee.org/images/files/pdf/iot-developer-survey-2016-report-final.pdf>.
- Dr. MD Rashedul Islam and Tridib Mazumder. Mobile application and its global impact. *International Journal of Engineering Technology*, 10(6):72–78, 01 2010.
- Vojtěch Jína. Javascript test runner. *examensarb., Czech Technical University in Prague*, 2013.
- William Jobe. Native apps vs. mobile web apps. *IJIM*, 7, 10 2013. doi: <http://dx.doi.org/10.3991/ijim.v7i4.3226>.
- Shravan Kumar Kasagoni. *Building Modern Web Applications Using Angular*. Packt Publishing Ltd, 2017.
- Norliza Katuk, Ku Ku-Mahamud, Nur Haryani Zakaria, and Mohd Maarof. Implementation and recent progress in cloud-based smart home automation systems. pages 71–77, 04 2018. doi: 10.1109/ISCAIE.2018.8405447.
- Tobias Krispinsson. *Hybrid application development : A comparison between native Android application and Ionic 2 application*. PhD dissertation, Harbin Institute of Technology, 2017.
- Somayya Madakam, R. Ramaswamy, and Siddharth Tripathi. Internet of things (iot): A literature review. *Journal of Computer and Communications*, 3(1):164–173, 2015. doi: [https://file.scirp.org/pdf/JCC\\_2015052516013923.pdf](https://file.scirp.org/pdf/JCC_2015052516013923.pdf).
- Kayla Matthews. The internet of robotic things: How iot and robotics tech are evolving together, 2019. URL <https://iot.eetimes.com/the-internet-of-robotic-things-how-iot-and-robotics-tech-are-evolving-together/>.
- Marco Nascimento. *Análise Comparativa de Protocolos em Smart Home: Considerações em Conectividade*. PhD dissertation, Universidade Federal de Santa Catarina - UFSC, 2016.
- R. Nunkesser. Beyond web/native/hybrid: A new taxonomy for mobile app development. In *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*, pages 214–218, 2018.

- Afif Osseiran, Omar Elloumi, JaeSeung Song, and Jose Monserrat. Internet of things. *IEEE Communications Standards Magazine*, 1(2):84–84, 2017.
- Keyur Patel, Sunil Patel, P. Scholar, and Carlos Scholar. Internet of things-iot: Definition, characteristics, architecture, enabling technologies, application future challenges. *IJESC*, 6(5), 2016. doi: [https://www.researchgate.net/publication/330425585\\_Internet\\_of\\_Things-IOT\\_Definition\\_Characteristics\\_Architecture\\_Enabling\\_Technologies\\_Application\\_Future\\_Challenges](https://www.researchgate.net/publication/330425585_Internet_of_Things-IOT_Definition_Characteristics_Architecture_Enabling_Technologies_Application_Future_Challenges).
- Suzanne Robertson and James Robertson. *Mastering the Requirements Process: Getting Requirements Right*. Addison-Wesley, 2013.
- Elar Saks. Javascript frameworks: Angular vs react vs vue. 2019.
- Shadi Sarawi, Mohammed Anbar, Kamal Alieyan, and Mahmood Alzubaidi. Internet of things (iot) communication protocols : Review. 07 2017. doi: 10.1109/ICITECH.2017.8079928.
- Elmustafa Sayed Ali Ahmed and Zeinab Kamal. Internet of things applications, challenges and related future technologies. *world scientific news*, 67(2), 01 2017.
- Eclipse SmartHome. A flexible framework for your smart home, 2019a. URL <https://www.eclipse.org/smarthome/index.html>.
- Eclipse SmartHome. Documentation, 2019b. URL <https://www.eclipse.org/smarthome/documentation/index.html>.
- Statista. Internet of things - number of connected devices worldwide 2015-2025, 2018. URL <https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/>.
- Statista. Number of apps available in leading app stores, 2020. URL <https://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/>.
- Sayali Sunil Tandel and Abhishek Jamadar. Impact of progressive web apps on web app development. *International Journal of Innovative Research in Science, Engineering and Technology*, 7(9):9439–9444, 09 2018.
- International Communication Unit I T U. Recommendation itu-t y.2060, 2012. URL <http://handle.itu.int/11.1002/1000/11559>.
- T. Vilček and T. Jakopec. Comparative analysis of tools for development of native and hybrid mobile applications. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1516–1521, 2017. doi: 10.23919/MIPRO.2017.7973662.

Christina Warren. Zuckerberg's biggest mistake? 'betting on html5, 2012. URL <https://mashable.com/2012/09/11/html5-biggest-mistake>.

Andrea Zanella, Nicola Bui, Angelo Castellani, Lorenzo Vangelista, and Michele Zorzi. Internet of things for smart cities. *IEEE*, 1(1):22–32, 2014.



---

## ANEXOS

---

### A.1 ESPECIFICAÇÃO DOS CASOS DE USO

|                             |   |
|-----------------------------|---|
| <b>Nome do Caso de Uso:</b> | Registo de um Utilizador  |
| <b>Descrição:</b>           | Permite que novos utilizadores se registem no sistema   |
| <b>Requisito Funcional:</b> | 1   |
| <b>Prioridade:</b>          | Alta  |
| <b>Actores:</b>             | Utilizador / Sistema  |
| <b>Condição de Entrada:</b> | O utilizador selecciona a interface de Registo  |
| <b>Fluxo Principal:</b>     | 1. O utilizador insere os dados da sua nova conta<br>2. O sistema verifica a informação inserida<br>3. O novo utilizador é automaticamente autenticado                |
| <b>Fluxo Alternativo:</b>   | 2a. O e-mail inserido não apresenta o formato correto<br>2b. O email inserido já se encontra registado no sistema<br>2c. As passwords inseridas não combinam entre si |

Tabela 15: Especificação do Caso de Uso: Registo de um Utilizador

|                             |  |
|-----------------------------|--|
| <b>Nome do Caso de Uso:</b> | Autenticação de um utilizador                            |
| <b>Descrição:</b>           | Permite que novos utilizadores se autentiquem no sistema |

|                             |  |
|-----------------------------|--|
| <b>Requisito Funcional:</b> | 2  |
| <b>Prioridade:</b>          | Alta   |
| <b>Actores:</b>             | Utilizador / Sistema   |
| <b>Condição de Entrada:</b> | O utilizador selecciona a interface de Login   |
| <b>Fluxo Principal:</b>     | <ol style="list-style-type: none"> <li>1. O utilizador insere os dados da sua conta</li> <li>2. O sistema verifica a informação inserida</li> <li>3. O utilizador é autenticado</li> </ol> |
| <b>Fluxo Alternativo:</b>   | <ol style="list-style-type: none"> <li>2a. O e-mail inserido não apresenta o formato correto</li> <li>2b. Os dados de autenticação estão incorretos</li> </ol>                             |

Tabela 16: Especificação do Caso de Uso: Autenticação de um utilizador

|                             |   |
|-----------------------------|---|
| <b>Nome do Caso de Uso:</b> | Adição de uma casa  |
| <b>Descrição:</b>           | Os utilizadores adicionam as suas casas no sistema  |
| <b>Requisito Funcional:</b> | 3   |
| <b>Prioridade:</b>          | Alta  |
| <b>Actores:</b>             | Utilizador / Sistema  |
| <b>Condição de Entrada:</b> | O utilizador o botão "Adiciona Casa"  |
| <b>Fluxo Principal:</b>     | <ol style="list-style-type: none"> <li>1. O utilizador insere os dados da sua nova casa</li> <li>2. O sistema verifica a informação inserida</li> <li>3. A casa é instalada no sistema</li> </ol>   |
| <b>Fluxo Alternativo:</b>   | <ol style="list-style-type: none"> <li>1a. O utilizador adiciona uma foto à sua nova casa através da câmara</li> <li>1b. O utilizador adiciona uma foto à sua nova casa através da galeria</li> <li>2a. O endereço do tunnel não apresenta o formato correto</li> </ol> |

Tabela 17: Especificação do Caso de Uso: Adição de uma casa

|                             |  |
|-----------------------------|--|
| <b>Nome do Caso de Uso:</b> | Adição de uma divisão  |
| <b>Descrição:</b>           | Os utilizadores adicionam divisões às suas casas   |
| <b>Requisito Funcional:</b> | 4  |
| <b>Prioridade:</b>          | Alta   |
| <b>Actores:</b>             | Utilizador / Sistema   |
| <b>Condição de Entrada:</b> | O utilizador carrega no botão "Adicionar Divisão"  |
| <b>Fluxo Principal:</b>     | 1. O utilizador selecciona a divisão que pretende inserir<br>2. A divisão é instalada no sistema |
| <b>Fluxo Alternativo:</b>   | 1a. Caso a divisão instalada seja um Quarto, o utilizador pode adicionar o seu proprietário      |

Tabela 18: Especificação do Caso de Uso: Adição de uma divisão

|                             |   |
|-----------------------------|---|
| <b>Nome do Caso de Uso:</b> | Adição de um dispositivo  |
| <b>Descrição:</b>           | Os utilizadores adicionam dispositivos às suas divisões   |
| <b>Requisito Funcional:</b> | 5   |
| <b>Prioridade:</b>          | Alta  |
| <b>Actores:</b>             | Utilizador / Sistema  |
| <b>Condição de Entrada:</b> | O utilizador carrega no botão "Adicionar Dispositivo"   |
| <b>Fluxo Principal:</b>     | 1. O introduz os dados referentes ao dispositivo<br>2. O sistema procura o dispositivo indicado pelo utilizador<br>3. O utilizador adiciona o dispositivo encontrado pelo sistema |
| <b>Fluxo Alternativo:</b>   | 2a. O sistema não encontra nenhum dispositivo referente à informação inserida   |

Tabela 19: Especificação do Caso de Uso: Adição de um dispositivo

|                             |  |
|-----------------------------|--|
| <b>Nome do Caso de Uso:</b> | Adição de um cenário de utilização   |
| <b>Descrição:</b>           | Os utilizadores adicionam cenários de utilização às suas casas   |
| <b>Requisito Funcional:</b> | 6  |
| <b>Prioridade:</b>          | Alta   |
| <b>Actores:</b>             | Utilizador / Sistema   |
| <b>Condição de Entrada:</b> | O utilizador carrega no botão "Adicionar Cenário"  |
| <b>Fluxo Principal:</b>     | <ol style="list-style-type: none"> <li>1. O utilizador selecciona o ícone para o cenário</li> <li>2. O utilizador define um nome para o cenário</li> <li>3. O sistema adiciona o cenário ao sistema</li> </ol> |
| <b>Fluxo Alternativo:</b>   |  |

Tabela 20: Especificação do Caso de Uso: Adição de um cenário de utilização

|                             |   |
|-----------------------------|---|
| <b>Nome do Caso de Uso:</b> | Adição de dispositivos a um cenário   |
| <b>Descrição:</b>           | Os utilizadores adicionam dispositivos pré-configurados no sistema aos seus cenários de utilização  |
| <b>Requisito Funcional:</b> | 7   |
| <b>Prioridade:</b>          | Alta  |
| <b>Actores:</b>             | Utilizador / Sistema  |
| <b>Condição de Entrada:</b> | O utilizador carrega no botão "Adicionar Dispositivos", enquanto visualiza um cenário de utilização |



|                           |   |
|---------------------------|---|
| <b>Fluxo Principal:</b>   | <ol style="list-style-type: none"> <li>1. O utilizador selecciona o dispositivo configurado a adicionar</li> <li>2. O utilizador define o estado que pretende que dispositivo possua</li> </ol> |
| <b>Fluxo Alternativo:</b> |   |

Tabela 21: Especificação do Caso de Uso: Adição de dispositivos a um cenário

|                             |  |
|-----------------------------|--|
| <b>Nome do Caso de Uso:</b> | Adição de uma tarefa cronometrada  |
| <b>Descrição:</b>           | Os utilizadores adicionam tarefas cronometradas às suas casas  |
| <b>Requisito Funcional:</b> | 8  |
| <b>Prioridade:</b>          | Alta   |
| <b>Actores:</b>             | Utilizador / Sistema   |
| <b>Condição de Entrada:</b> | <ol style="list-style-type: none"> <li>1. O utilizador selecciona o separador de "Tarefas Cronometradas"</li> <li>2. O utilizador carrega no botão "Adicionar Tarefa"</li> </ol>   |
| <b>Fluxo Principal:</b>     | <ol style="list-style-type: none"> <li>1. O utilizador selecciona o dispositivo ou cenário a aplicar na tarefa</li> <li>2. O utilizador selecciona o período em que deseja aplicar a tarefa</li> <li>3. O sistema regista a nova tarefa e cria um tarefa assíncrona (delayed job)</li> </ol> |
| <b>Fluxo Alternativo:</b>   |  |

Tabela 22: Especificação do Caso de Uso: Adição de uma tarefa cronometrada

|                             |                               |
|-----------------------------|-------------------------------|
| <b>Nome do Caso de Uso:</b> | Adição de uma tarefa acionada |
|-----------------------------|-------------------------------|

|                             |   |
|-----------------------------|---|
| <b>Descrição:</b>           | Os utilizadores adicionam tarefas acionadas às suas casas   |
| <b>Requisito Funcional:</b> | 9   |
| <b>Prioridade:</b>          | Alta  |
| <b>Actores:</b>             | Utilizador / Sistema  |
| <b>Condição de Entrada:</b> | <ol style="list-style-type: none"> <li>1. O utilizador selecciona o separador de "Tarefas Acionadas"</li> <li>2. O utilizador carrega no botão "Adicionar Tarefa"</li> </ol>  |
| <b>Fluxo Principal:</b>     | <ol style="list-style-type: none"> <li>1. O utilizador selecciona o dispositivo ou cenário a aplicar na tarefa</li> <li>2. O utilizador selecciona a tarefa que irá acionar a tarefa</li> <li>3. O utilizador define o estado que o respetivo dispositivo deverá assumir para acionar a tarefa</li> <li>4. O sistema regista a nova tarefa e cria um tarefa assíncrona (delayed job)</li> </ol> |
| <b>Fluxo Alternativo:</b>   |   |

Tabela 23: Especificação do Caso de Uso: Adição de uma tarefa acionada

|                             |  |
|-----------------------------|--|
| <b>Nome do Caso de Uso:</b> | Consulta da meteorologia   |
| <b>Descrição:</b>           | Os utilizadores visualizam a meteorologia para a cidade pré-definida |
| <b>Requisito Funcional:</b> | 30   |
| <b>Prioridade:</b>          | Média  |
| <b>Actores:</b>             | Utilizador / Sistema   |
| <b>Condição de Entrada:</b> | 1. Na homepage utilizador desliza para o ecrã da meteorologia        |

|                           |  |
|---------------------------|--|
| <b>Fluxo Principal:</b>   | <ol style="list-style-type: none"> <li>1. O sistema obtém a cidade do utilizador</li> <li>2. O sistema invoca a API de meteorologia utilizando a cidade do utilizador</li> <li>3. O sistema apresenta a informação obtida</li> </ol> |
| <b>Fluxo Alternativo:</b> | 3a - Caso ocorra um erro na invocação da API, não é apresentada qualquer informação  |

Tabela 24: Especificação do Caso de Uso: Consulta da meteorologia

|                             |  |
|-----------------------------|--|
| <b>Nome do Caso de Uso:</b> | Consulta de estatísticas   |
| <b>Descrição:</b>           | Os utilizadores visualizam estatísticas relativas às suas divisões, casas, dispositivos, cenários de utilização e tarefas  |
| <b>Requisito Funcional:</b> | 31   |
| <b>Prioridade:</b>          | Média  |
| <b>Actores:</b>             | Utilizador / Sistema   |
| <b>Condição de Entrada:</b> | 1. Na homepage utilizador selecciona as estatísticas que deseja visualizar   |
| <b>Fluxo Principal:</b>     | <ol style="list-style-type: none"> <li>1. O utilizador selecciona o dispositivo/cenário/tarefa sobre o qual deseja visualizar as estatísticas</li> <li>2. O sistema apresenta as respectivas estatísticas ao utilizador</li> </ol> |
| <b>Fluxo Alternativo:</b>   |  |

Tabela 25: Especificação do Caso de Uso: Consulta de estatísticas

