



Vadson
Guilherme
Avelino
Culanda

SmartHome - Desenvolvimento de App Mobile em
Flutter para gestão da casa inteligente
SmartHome-Mobile App Development in Flutter
for smart home management





**Vadson
Guilherme
Avelino
Culanda**

**SmartHome - Desenvolvimento de App Mobile em
Flutter para gestão da casa inteligente
SmartHome-Mobile App Development in Flutter
for smart home management**

Dissertação apresentada à Universidade de Aveiro para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrónica e Telecomunicações, realizada sob a orientação científica de Paulo Jorge de Campos Bartolomeu, Professor Adjunto da Escola Superior de Tecnologia e Gestão de Águeda

FCT Fundação
para a Ciência
e a Tecnologia

Este trabalho é financiado pela FCT/MCTES através de fundos nacionais e quando aplicável cofinanciado por fundos comunitários no âmbito do projeto UIDB/50008/2020-UIDP/50008/2020.

o júri / the jury

presidente / president

Professor Doutor António José Ribeiro Neves

Professor Auxiliar da Universidade de Aveiro (por delegação da Reitora da Universidade de Aveiro)

vogais / examiners committee

Professor Doutor Mário João Barata Calha

Professor Auxiliar da Universidade de Lisboa (Arguente Principal)

Professor Doutor Paulo Jorge de Campos Bartolomeu

Professor Adjunto Convidado da Universidade de Aveiro (Orientador)

agradecimentos / acknowledgements

É com muito gosto que aproveito esta oportunidade para agradecer a todos os que ajudaram-me durante o percurso académico. Antes de mais, gostaria de agradecer aos meus pais, aos meus irmãos e a minha tia, que acompanharam tudo de perto e sempre me apoiaram.

Quero também agradecer a Inova-Ria e a Altice labs, pela oportunidade de realizar a minha dissertação em contexto empresarial, especialmente ao meu orientador Herlander Santos, que garantiu as melhores condições de trabalho, atribuiu-me diversas tarefas que ajudaram a ganhar conhecimento e experiência, e introduziu-me a uma equipa excelente que auxiliou durante este projeto. Tal como, o João Ferraz que coorientou o meu trabalho, o Bruno Antunes que guiou-me pelo mundo do Flutter, partilhando a sua experiência e oferecendo dicas sobre as melhores práticas, o Vasco Coutinho que me introduziu ao mundo do teste de produto, de modo geral a equipa toda que tornou este projeto possível.

Também gostaria de agradecer o meu orientador da Universidade de Aveiro, o Prof. Doutor Paulo Bartolomeu que sempre supervisionou atentamente o meu trabalho, proporcionou-me todas as ferramentas necessárias e esteve sempre disponível a ajudar-me.

Por último, gostaria de agradecer a todos os amigos incríveis que fiz durante o meu percurso, são muitos para enumerar, mas sabem quem são. Desde amigos do curso, à amigos de Erasmus, à amigos da ESN e àqueles que já me conheciam antes de entrar na universidade. Muito obrigado a todas as pessoas mencionadas, que foram muito importantes neste capítulo da minha vida.

Palavras-chave

casas inteligentes, automação da casa, assistentes virtuais, dispositivos inteligentes, programação, indústria

Resumo

A *Internet das Coisas* acelerou a disponibilização de soluções para casas inteligentes, oferecendo uma gama cada vez mais variada de produtos e de aplicações de domótica inteligente que visam simplificar as tarefas do dia-a-dia e aumentar o conforto dos utilizadores. Contudo, frequentemente os sistemas disponibilizados são excessivamente complexos na perspectiva do utilizador, tornando a sua utilização pouco intuitiva e imprevisível. No âmbito desta dissertação foi desenvolvida uma aplicação para casas inteligentes baseada na framework *Flutter* que é capaz de interagir com o ecossistema de dispositivos Smart Home da Altice labs. Este documento apresenta uma comparação entre as diversas soluções de casas inteligentes existentes, além da visão geral da solução desenvolvida pela Altice labs. A implementação da solução proposta é detalhada recorrendo a excertos de código que ilustram as opções tomadas em termos de design e de funcionalidade. Por fim, a aplicação desenvolvida é avaliada recorrendo a um processo de teste que verifica o cumprimento dos seus requisitos.

Keywords

smart homes, home automation, virtual assistants, smart devices, programming, industry

Abstract

The *Internet of Things* has accelerated the availability of solutions for smart homes, offering an increasingly varied range of products and smart home applications that aim to simplify daily tasks and increase user comfort. However, the available systems are often excessively complex from the user's perspective, making their use unintuitive and unpredictable. Within the scope of this dissertation, an application for smart homes was developed based on the Flutter *framework*, which can interact with Altice labs' Smart Home device ecosystem. This document presents a comparison between the various existing smart home solutions, as well as an overview of the solution developed by Altice labs. The implementation of the proposed solution is detailed using code excerpts that illustrate the choices made in terms of design and functionality. Finally, the developed application is evaluated using a test process that verifies compliance with its requirements.

Contents

Contents	i
List of Figures	iii
List of Tables	v
1 Introduction	1
1.1 Focus and Scope	1
1.2 Objectives	2
1.3 Outline	2
2 State of the art	3
2.1 Tecnologies	3
2.2 Major Functionalities and Use Cases	3
2.2.1 Health	3
2.2.2 Security	4
2.2.3 Entertainment	4
2.2.4 Sustainability	4
2.3 Virtual assistants	4
2.3.1 Amazon Alexa	4
2.3.2 Google Home Assistant	8
2.4 Smart Home solutions	11
2.4.1 SmartThings Samsung	11
2.4.2 Apple Homekit	14
2.4.3 TuyaSmart	17
2.5 Which Solution is better	20
2.5.1 Compatible Devices	20
2.5.2 Digital Assistants	20
2.5.3 Automation	21
2.5.4 Extra features	22
2.5.5 Conclusion	22
3 Altice Home app	23
3.1 Introduction	23
3.2 Objectives	23
3.3 Innovations aspects	24
3.4 Tools	24

3.5	Devices	24
3.6	Architecture	25
3.6.1	Cloud	25
3.6.2	APIs (Outsourced)	26
3.6.3	Tuya SDKs(Outsourced)	26
3.6.4	Flutter Mobile App for Android and iOS	26
4	Design and Implementation	29
4.1	Application Design	32
4.2	Implementation	32
4.3	Login Screen Design	33
4.4	Login Screen Implementation	35
4.5	Room Screen Design	37
4.5.1	Main Screen	37
4.5.2	Add New Room	39
4.5.3	Choose Devices	40
4.6	Room Screen Implementation	41
4.6.1	Main Screen Implementation	41
4.6.2	Add New Room Implementation	45
4.6.3	Choose Devices Implementation	51
4.7	Room Screen 1st Implementation	53
4.7.1	Main Screen 1st Implementation	53
4.7.2	Add New Room 1st Implementation	56
4.8	User Account Screen Design	61
4.9	User Account Screen Implementation	63
4.10	Settings Screen Design	65
4.11	Settings Screen Implementation	67
5	Testing	69
5.1	What is testing?	69
5.2	Testing Throughout the Software Development Lifecycle	69
5.2.1	Test levels	69
5.3	Test management tool	70
5.4	Test Plan	71
5.4.1	Regression tests	71
5.4.2	Acceptance tests	74
5.4.3	Exploratory tests	74
5.5	Validation	75
6	Conclusion	79
6.1	Conclusion	79
6.2	Future Works	81
	Bibliography	83

List of Figures

2.1	(a) Home (b) Communication (c) Entertainment	7
2.2	(a) Devices (b) More	8
2.3	Home Assistant dashboard-Lovelace	11
2.4	(a) Introduction (b) Home (c) Panel (d) Add device	14
2.5	(a) Introduction (b) Home (c) Automation (d) Add device	16
2.6	(a) Login (b) Home (c) Remote Control	19
2.7	(a) Instant Connection (b) One tap to share	20
3.1	Global architecture	25
3.2	Diagram of process of pairing in Wi-Fi EZ mode by Tuya Inc.	26
3.3	(a) Home (b) Home menu (c) Add device (d) Device info	27
3.4	(a) Add room (b) Add scenario	28
4.1	Use cases diagram	29
4.2	Functional and Non-Functional Requirements	31
4.3	Functional and Non-Functional Requirements	31
4.4	Text field factory	33
4.5	Login Screen	34
4.6	Login Screen with labels	35
4.7	Login Screen code excerpt	36
4.8	Login Screen code excerpt	36
4.9	Room Main screen	38
4.10	Add new room screen	39
4.11	Choose Devices screen	40
4.12	Main Screen with labels	41
4.13	RoomPage excerpt	42
4.14	RoomCard code excerpt	43
4.15	RoomsPageCubit code excerpt	44
4.16	RoomsPageCubit code excerpt	44
4.17	Room model constructor code excerpt	45
4.18	Add New Room Screen with labels	46
4.19	Add New Room Page excerpt	47
4.20	Add New Room Page excerpt	47
4.21	RoomTypeSquareButton class code excerpt	48
4.22	utils.getSVGIllustrationPathforRoom code excerpt	48
4.23	Add New Room Page excerpt	49

4.24	Add New Room Page excerpt	49
4.25	Add New Room Page Cubit code excerpt	50
4.26	Choose Devices screen with labels	51
4.27	RoomPage excerpt	53
4.28	RoomPage excerpt	53
4.29	RoomPage excerpt	54
4.30	RoomPage excerpt	54
4.31	RoomsPageCubit code excerpt	55
4.32	RoomsPageCubit code excerpt	55
4.33	Add New Room Page code excerpt	56
4.34	Add New Room Page code excerpt	56
4.35	Add New Room Page code excerpt	57
4.36	Add New Room Page excerpt	58
4.37	Add New Room Page excerpt	58
4.38	Add New Room Page excerpt	59
4.39	Add New Room Page excerpt	60
4.40	Add New Room Page excerpt	60
4.41	Add New Room Page Cubit code excerpt	61
4.42	User Account Screen	62
4.43	UserAccount Screen with labels	63
4.44	UserAccountPage excerpt	64
4.45	UserAccountCubit excerpt	64
4.46	UserAccountModel code excerpt	64
4.47	UserAccountRepository code excerpt	65
4.48	Settings Screen	66
4.49	SettingsItem code excerpt	67
4.50	SettingsCubit code excerpt	67
5.1	Xray layout	71
5.2	Regression testing	72
5.3	Regression testing	72
5.4	Regression testing	73
5.5	Table of tests executed in all of the screens	75
5.6	Table of test executed for the "Add New Room" screen	75
5.7	Table of test executed for the "Choose Devices" screen	76
5.8	Code review of a pull request	77

List of Tables

2.1 Digital assistants features 21

Chapter 1

Introduction

1.1 Focus and Scope

Nowadays everything around us is becoming smart, our cars, our smartphones, and what about our houses? With the technological advancement and the need for comfort and simplicity, a huge market opened for the smart home.

Smart homes can be viewed as independent houses. They can be controlled through various devices which are programmed to execute specific tasks and functions. Originally, Smart home technologies were simply used to control lighting and heating systems. Now almost every electric device can be included in the Smart Home ecosystem to allow monitoring the external environment as well as all other activities happening inside the house[1]. Modeling the behavior of the devices according to a set of patterns or predefined scenarios, house customization based on the inclusion of various devices to different divisions, and scheduling of tasks are many of the existing possibilities. This kind of control can be done through countless Smart home apps or virtual assistants for smartphones or tablets. From all the available solutions we can highlight, Google Home assistant, Amazon Alexa, SmartThings, Apple Homekit, and Tuya Smart.

This dissertation was conducted in a business environment. The smart home project is being developed by Altice labs and has a goal to produce an innovative solution that can rival existing ones. Doing the dissertation in this context is analogous to doing an internship which brings a lot of benefits. Being part of an app development helps understanding how the job market works, and gives an insight into all the phases that a project undergoes. The first part is the UI design and the MVP (Minimum Valuable Product) proposition, after the design is completed then we move to the development part which operates in parallel with the testing and validation. Finally, we move to the release and the beta phase where we choose a select group of volunteers to test the product and give feedback.

Altice labs is a company that continuously invests in innovation. They develop equipment and solutions in areas such as Medicine, Education, Culture, and Sports. Their main concern rests in ensuring the best customer experiences by creating added value in every client segment and continuously engaging in collaborative research and development projects as part of a sustained technological leadership strategy. Exploring innovation activities around strategic themes such as Artificial Intelligence Machine Learning, Cloud technologies (computing and

networking), Smart Living, Internet of Things, Big Data, Security Privacy, Digital Services Platform, 5G and Future Networks, including the Evolutionary Optical Framework.

1.2 Objectives

As stated previously the main goal of this project is the development in Flutter of a mobile app for smart home management whose vision is to achieve a complete end-to-end solution that allows the management of a smart home in an integrated and easy way. As specific goals we can highlight:

- Research and analyze existing smart home solutions
- Become familiar with flutter
- Implement screens related to the solution
 - Login screen
 - Room screen
 - User account screen
 - Settings screen
- Validate the solution by executing use case tests

1.3 Outline

Until now we introduced the context of this dissertation, the description of the problem, and its relevance. The remainder of this dissertation is organized as follows:

- **Chapter 2-** This chapter provides an overview and comparison between existing smart home solutions.
- **Chapter 3-** This chapter presents an overview of the solution, Altice Home App, as well as a global vision of the app, major functionalities, and the chosen platform for implementation
- **Chapter 4 and 5-** These chapters highlight the individual work done as part of the global solution developed by the Altice labs and the validation of the same
- **Chapter 6-** chapter presents the main conclusions of this work along with the future work perspectives.

Chapter 2

State of the art

2.1 Technologies

In an initial phase, smart home automation was focused on the installation of plugs or light switches, and the wiring of infrared controllers all over the house [2]. With the development of the Internet, mobile communications, and renewable energy technologies came a significant improvement in smart homes, particularly in better house infrastructures, equipment, and added motivation for investments related to domotics and control technologies. Connectivity is a very important aspect in the development of smart homes since almost every device is dependent on this factor. Bad connectivity implies not only frustrations for the user but also a huge risk to the security of the system [3]. Therefore, the fast development of mobile communications has caused a great improvement in this technology. Wireless networks and smart devices with wireless interface communications (e.g., Bluetooth, ZigBee, Wi-Fi) are omnipresent and provide a better experience to the user [2]. Currently, smart homes have a high impact in terms of comfort, security, and energy savings for their users. With the development of cloud-based solutions, users have gained access to remote control of every resource, configuration, and notification of one or more smart homes. And with voice commands, a better-personalized experience can be obtained through digital assistants that can be called by name.

2.2 Major Functionalities and Use Cases

Smart homes have the goal of simplifying the lives of their residents, saving energy and providing comfort and security. They can be used in areas related to health, safety, entertainment, and sustainability.

2.2.1 Health

In this field, the goal is to provide a better quality of life and make sure that the elderly community can live comfortably and independently. Multiple devices can be used in this area, for example, a smart wristband can be used to monitor someone's vital signs and control if any anomalies are registered, and in case they are, it immediately notifies a health assistant. An air purifier can also be used at home to provide better living conditions, which are beneficial not only for people with asthma but to everyone.

2.2.2 Security

Many solutions can be implemented like intruders detection, monitoring throughout security cameras, panic buttons, and danger prevention and detection, namely carbon monoxide, fire,...etc.

2.2.3 Entertainment

Several devices can provide improved entertainment experiences, for example, with the help of a smart TV, smart light, and smart speaker, we can obtain an amazing personalized audiovisual experience.

2.2.4 Sustainability

Allows energy-saving and decrease of water consumption, through automatic irrigation systems, automatic lights, temperature control , among other options.

2.3 Virtual assistants

2.3.1 Amazon Alexa

Description

Alexa is a virtual assistant that supports a Smart Home solution developed by Amazon, which allows the user to realize multiple tasks, such as creation of lists, getting the weather conditions, access to articles available on the Internet, among many others. Alexa has an integrated voice assistant that can answer questions and execute functions or skills. Thanks to the conversion of the sound waves to text by Alexa, it allows the Wolfram Language technology to process the data and generate precise and accurate answers from multiple sources [4].

Smart Devices

- Lights - Ability to vary the intensity of the light or change the color to adjust the environment of each room to the user's preference.
- Plugs - Allow standard appliances to be integrated into the smart home experience, such as lamps, Christmas lights, fans, and more.
- Thermostats - They have many economic and environmental benefits. By regulating the temperature of the house when the user is absent, allowing energy savings.
- Cameras - They have many applications in terms of security, such as monitoring children and pets and preventing intruders.

Device Communication

Communication between devices and Alexa can be done in any of the following ways:

- Alexa Connect Kit cloud services

- Connecting directly to Alexa’s local service through the ZigBee protocol (2.4GHz).
- Through a secondary device, such as a hub, controllable using an Alexa skill provided by the hub manufacturer.
- Through a pre-certified IoT devices from a service provider, such as Tuya

Home Automation

Skills

The skills work as add-ons. Each one enriches Alexa more and allows it to execute several new features. For example, didactic skills such as the Jeopardy game, auditory skills such as the Anypod podcast [5], or even skills to control the home, such as the case of Tuya Smart, which allows through simple voice commands to control various smart devices that we have at home.

How do skills work?

Skills are composed of two strands, an interactive model- or voice interface- and application logic. When a customer speaks, Alexa analyzes the context of the interactive model to determine the customer’s order. Then it sends the request to the application logic, which will execute it. The application logic is given as a back-end cloud service where the host is Alexa, AWS, or any other server.

How to set up a Skill?

Developers use Amazon Web Services (AWS) Lambda, just enter Lambda’s Amazon Resource Name (ARN) into the skill configuration. AWS Lambda is compatible with Node.js (JavaScript), Java, Python, C # or Go.

Scenarios

Users can use voice commands to configure multiple devices with pre-defined configurations. For example, the user could say “Alexa, activate bedtime” and the light intensity would decrease and the room temperature would decrease. The user can create and edit scenarios through any Alexa compatible hub or the application of the device manufacturers.

Compatibility

Alexa is compatible with smart devices from various manufacturers, such as SNAS, Fibaro, Belkin, ecobee, Geeni, IFTTT, Insteon, LIFX, LightwaveRF, Nest, Philips Hue, SmartThings, Wink, and Yonomi.

IoT Solution Providers

They offer hardware modules, software, and services that help users pair their devices with Alexa. The providers that will be presented develop smart home skills for other companies as well [6].

- **Ayla Networks** - It works on AWS and includes voice skills developed for Alexa that can be integrated with customized mobile applications to control smart home devices, such as locks, lights, and temperature control.
- **Broadlink** - Setup is done without using any application, works on AWS, and includes a customizable application that helps to improve the efficiency of the product.
- **Coolkit e Welink** - It includes a firmware module that supports Wi-Fi, Zigbee, GSM, and Bluetooth. It also includes PCB hardware, a global software service platform and open API, customizable Android and IOS applications, and simple integration with Alexa.
- **Tuya** - It provides smart devices for various brands and manufacturers. Includes access to hardware, cloud services, and application development.

UI and Screen

In figures 2.1 and 2.2 we have an UI example of the Alexa smart home solution.

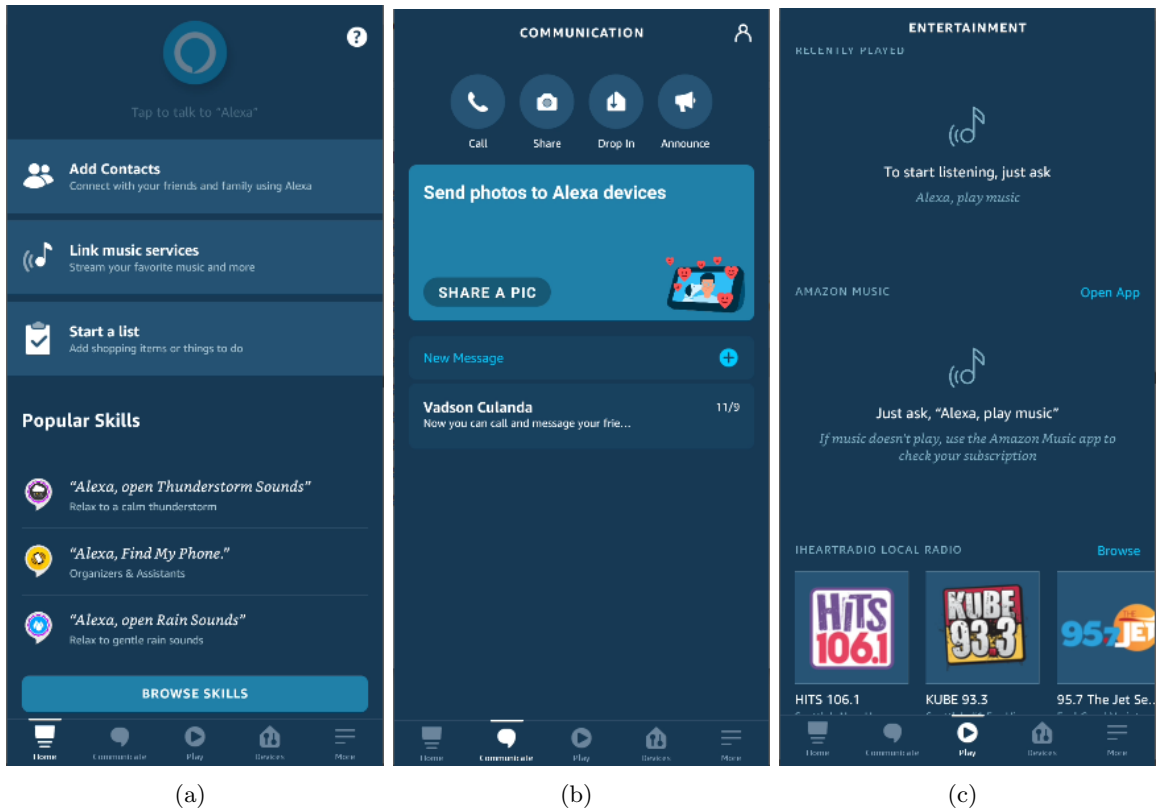


Figure 2.1: (a) Home (b) Communication (c) Entertainment

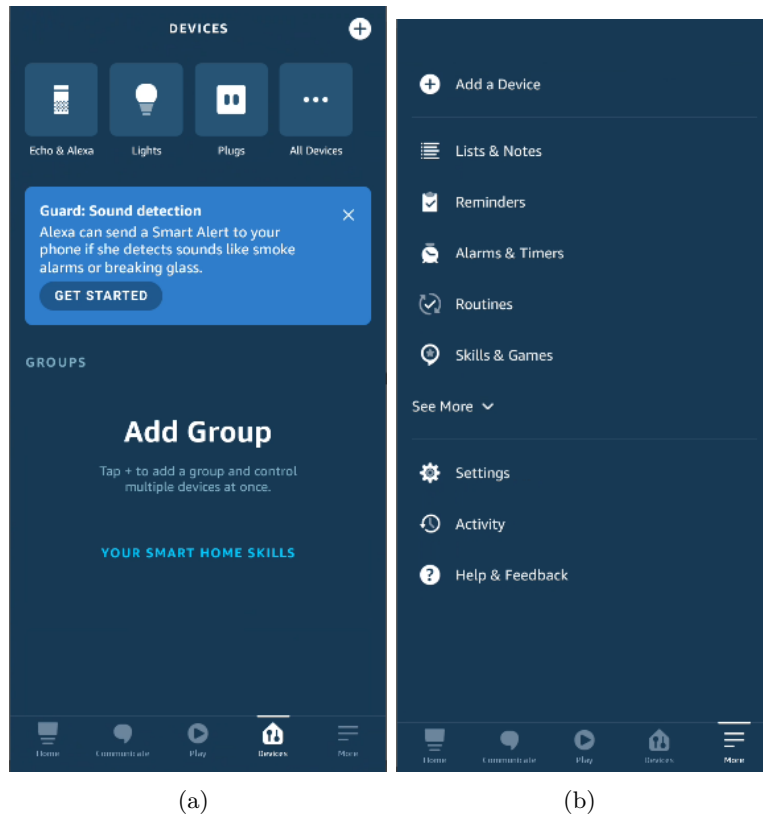


Figure 2.2: (a) Devices (b) More

2.3.2 Google Home Assistant

Description

It is a free open-source software, developed in Python, that allows the management and automation of smart homes. The Home Assistant Core is a program developed in Python that can be implemented on servers that use different operating systems [7]. It facilitates its execution on devices, such as a RaspberryPi, virtual machine, and other hardware platforms. The Home Assistant assumes the role of a hub controller that encompasses multiple Smart home functions, such as security alarm management, sensors for temperature measurement and control, light control, television, speakers. It also supports voice commands, mobile applications, and a user interface (front-end) available on the Home Assistant website. The Application's main focus is on local control and privacy. It supports smart devices for numerous brands and manufacturers, while including access to hardware, cloud services, and application development.

Device Communication

Z-wave

It is a wireless communication protocol developed for home automation. It uses a low power

network mesh that allows devices that are not in direct reach with each other, to communicate indirectly through nodes that allow the redirection of messages. Each node corresponds to a device that is powered by alternating current. Battery-powered devices such as door locks or some types of thermostats are unable to redirect messages[8]. To use this protocol, a Z-wave controller and one or more devices are required. This controller can be a Z stick or a hub that supports this protocol, from several hubs compatible with the Home Assistant we can identify **SmartThings, Fibaro, Wink, and Vera**. The efficiency of this protocol is directly related to network mesh, and this is more stable the more associated devices you have.

MQTT(MQ Telemetry Transport)

It is an IoT connectivity protocol in addition to TCP / IP. It has the advantage of requiring a very simple configuration on the side of the Home Assistant. The configuration of the devices is carried out directly on them. To avoid multiple entries of the same device (in the case of reconnections), each device is assigned a unique identifier [9]. MQTT has a function that allows to automatically find devices that support this protocol, including **alarm control panels, binary sensors, cameras, fans, lights, sensors, tags**, etc.

Home Automation

Triggers

Triggers are responsible for the automation process. When any of the triggers is activated, the Home Assistant validates the **conditions** and then executes the **action** (if any). There are different types of triggers, and for certain automation, there can be multiple triggers. Below are some examples of triggers [10]:

- **Event trigger** - is fired when an event is received. Events can be activated through the name, through some specific data, or depending on the context.
- **Home Assistant Trigger** - is triggered when the home assistant turns on or off.
- **State trigger** - is activated whenever the state of a given entity changes.

Conditions

Conditions are optional and can be used to prevent a certain action from being performed when a trigger occurs. Although they look similar, the conditions and triggers are very different. The conditions only observe the system discreetly while the triggers look continuously in time [11]. For example, a trigger can observe that a switch is being turned on but the condition only sees if the switch is on or off. “And”, “or”, “not” and “sunState condition” are some examples of conditions.

Actions

The actions are executed when a trigger is fired through “services” or “events”. In the case of services, the user can specify the device id that will be affected as an optional parameter (for

example to define the brightness). Services can also be used to activate a given scenario[12].

Scenes

The user can create scenes that define a specific state of an entity. For example, the user can define that Light A should be turned on and the color of Light B should be placed in red[13].

Compatibility

Home Assistant is compatible with smart devices from different manufacturers, such as Amazon Alexa, Apple HomeKit, Bluetooth, ecobee, Google Assistant, Google Cast (Google Chromecast), Google Home, Google Nest, IFTTT, IKEA Smart Home, KNX, Xiaomi Smart Home, MQTT, Philips Hue, SmartThings (Samsung), Tuya Smart, X10, Zigbee, Z-Wave, among others.

IoT Solution Providers

- **Smartthings (Samsung)** - It is integrated into the Home Assistant through the Cloud SmartThings API. Some of the features are as follows:
 - Control SmartThings devices as if they were Home Assistant entities.
 - Automatic synchronization of entities, which have changed in SmartThings, by restarting the Home Assistant.
 - There is no need for additional dependencies.
- **Zigbee Home Automation** - Allows direct integration into the Home Assistant of various devices compatible with Zigbee.
- **Philips Hue** - Allows to control and monitor the lights and motion sensors connected to the Hue bridge. Hue integration is done automatically, if not, the device can be added through the menu.
- **Ecobee**-Allows to control and view the data from the ecobee thermostat. For example, it allows to define a climate that is a set of predefined conditions that the thermostat will try to satisfy. The ecobee thermostat includes 3 predefined climates, “Home”, “Away” and “Sleep”.
- **Amazon Alexa** - With the help of the Home Assistant Cloud, integration with Alexa is quick and effective.
 - Allows you to control devices through Alexa routines.
 - Allows you to execute voice commands without having to know the name of the skill, for example, “Alexa turns off the light”.
 - Allows to view and control devices through the Alexa application.

UI and Screen

In terms of UI in addition to the mobile application, there is Lovelace (figure 2.3), which is a Home Assistant dashboard with the same features as the mobile application. Lovelace is customizable, fast, and is a very interesting and efficient way for users to control their homes. Access can be done on the Home Assistant website, via mobile phone, or computer. Below are some of the features of Lovelace:

- There are 27 different cards to add and configure to taste.
- The dashboard editor allows you to manage Lovelace and even includes a real-time forecast.
- The cards have several options for configuring how the data is displayed.
- Theme configuration.
- Access to an existing card repository.

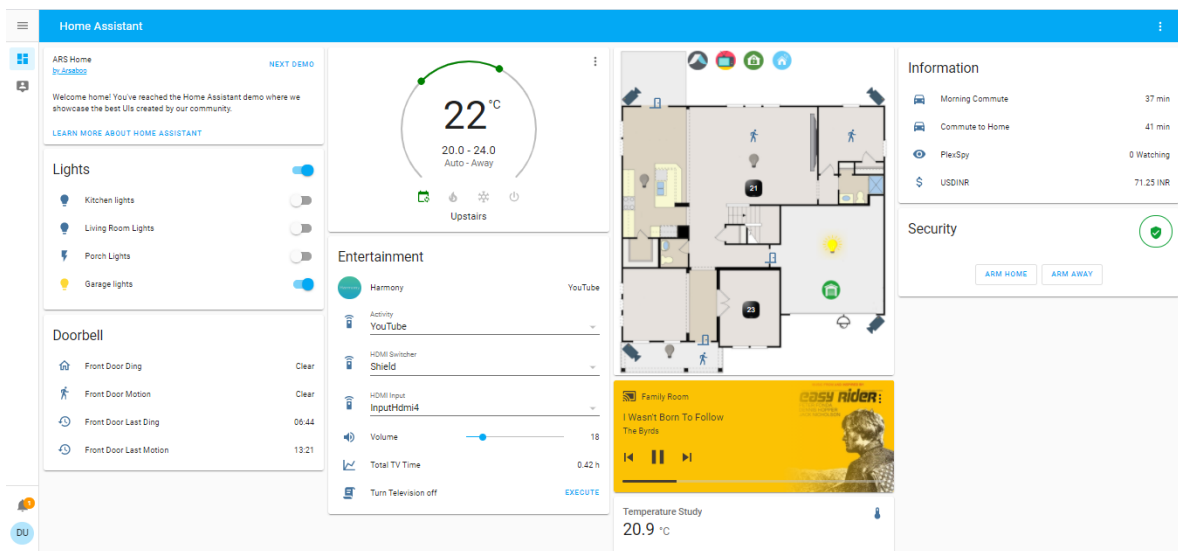


Figure 2.3: Home Assistant dashboard-Lovelace

2.4 Smart Home solutions

2.4.1 SmartThings Samsung

Description

SmartThings is a Smart Home solution developed by Samsung, which allows the user to make a wireless connection to a large number of smart devices, which can work simultaneously. SmartThings has an ecosystem called “The SmartThings ecosystem”, which allows you to create and integrate IoT devices, services, and automation in the SmartThings cloud [14]. This ecosystem includes the following components:

- **Devices** - can connect directly to the SmartThings cloud, or a SmartThings-compatible hub, via a third-party cloud.
- **Automation** - are WebHook or AWS Lambda functions that allow the user to control the ecosystem without manual intervention.
- **The SmartThings app** - is used to configure and control automation and IoT devices supported by SmartThings.
- **Bixby voice Assistant** - allows control and management of devices through voice.
- **Developer Workspace** - is a space with useful tools for adding IoT devices and automations to the SmartThings cloud.
- **The Smarthings API** - allows you to integrate, control, and monitor IoT devices and services from the SmartThings cloud.

Device Communication

Communication between SmartThings devices and the ecosystem can be done in any of the following ways [15]:

- Devices connected to the SmartThings cloud through third-party clouds communicate via:
 - SmartThings Schema Connector- is the fastest and most recommended way for clouds that support OAuth 2.0.
 - SmartApp Connector- is an option to implement certain advanced features or to customize the device’s onboarding Ux.
- Hub connected devices communicate through SmartThings and Zigbee or Z-Wave compatible hubs.
- Directly connected devices automatically connect to the SmartThings cloud. This type of device uses the MQTT Protocol.

Home Automation

Automations

An automation performs actions based on a set of conditions such as specific times, days of the week, or when a device is activated (for example, a motion sensor).

How do automations work?

Automations work through the “if” (IF) “then” condition (then) action flow. That is, “If” a certain condition happens, “then” controls the device, executes the scenario or the SmartApp, or controls the location [16, 17].

Scenes

The scenes allow you to control different actions of multiple devices with the touch of a button. The scenarios can be activated manually or automatically through automation. Unlike automations, scenes do not have triggers. Automation can be something like “notify me if the doors open and I’m not at home”. This automation will wait for that situation to occur to notify the user, it does not happen instantly. On the other hand, a scenario that is to “turn off all kitchen and living room lights” will be executed the moment it is activated (with the press of the button) [18].

SmartApps

SmartApps are pre-configured services that give special control over the devices. They can have a variety of functions - from lock management to activation of security events for the smart home. SmartApps are programs written in the Groovy language, similar to Java, and developed in the sandbox.

IoT Solution Provider

- **Hub** - is the brain of the house.
 - Aetoc Smart Home Hub
 - V-Home from the Vodafone Security Starter Kit.
- **Home builder and developers:**
 - Truland Homes
 - VolkerWessels
 - JM Residential
- **Property management** - Automates, monitors and manages the home with ease.
 - Microsoft

UI and Screen

In figure4 2.4 an UI example of the SmartThings smart home solution is shown.

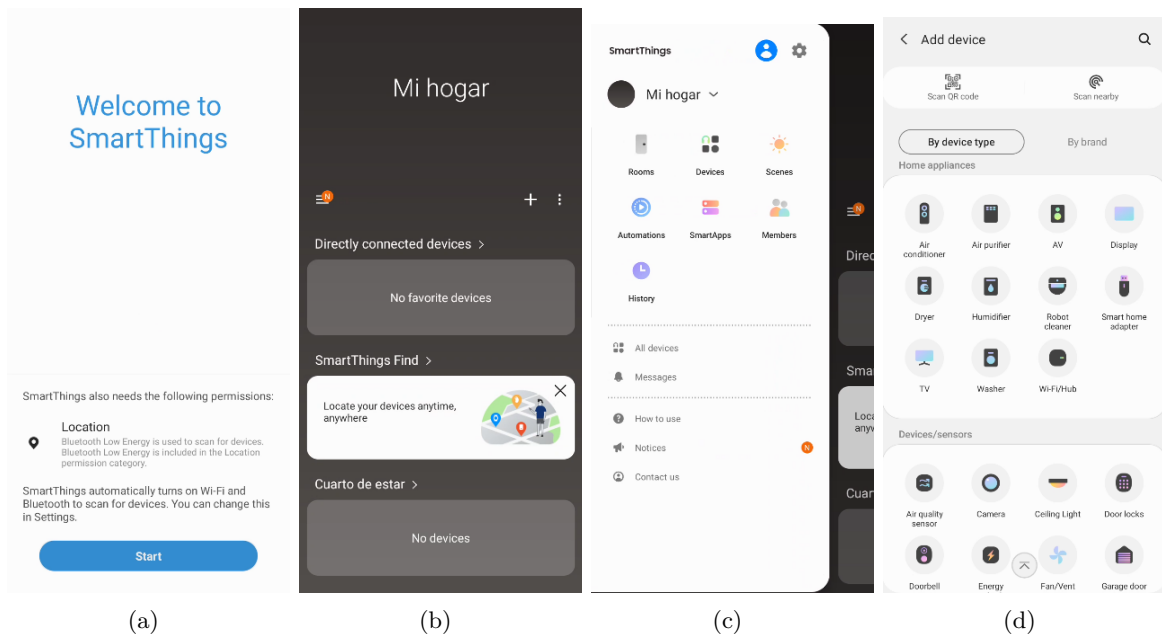


Figure 2.4: (a) Introduction (b) Home (c) Panel (d) Add device

2.4.2 Apple Homekit

Description

The Apple Homekit is a Smart Home solution developed by Apple, which allows users to control, manage and configure various devices, applications, and services connected to the Homekit Framework, through Siri or the Home app available on the iPhone, iPad, Apple Watch, and Mac [19, 20, 21].

Note: In Apple HomeKit “devices” are referred to as “accessories”.

Device Communication

Communication between devices is done through the HomeKit Accessory Protocol (HAP), this is a protocol that supports IP and Bluetooth LE communication. This communication is local, when the user is at home the iPhone communicates directly with the device, and when the user is away the iPhone communicates with a Home Hub (HomePod, Apple Tv, or an iPad) and the latter communicates with the device. This type of communication is advantageous in terms of security since only the Hub communicates with the cloud while the other devices communicate locally with each other [22]. The integration of the devices is done very simply through the use of the HomeApp to make a scanner with a 8-digit code that is contained in the device compatible with the HomeKit, either in its instruction manual or in the box. This integration can also be done through NFC since devices compatible with Homekit have this functionality [23].

Home Automation

Homes

Homekit uses the term “home” to represent a physical house, an office, or some other place that has some importance to the user. The user can have multiple homes[21, 24]

Rooms

A “room” represents a physical room in the house. The rooms are simply names that have meaning for people, for example, the living room and the attic. After having devices associated with the rooms, you can use various voice commands provided by Siri, such as “Siri, turn off all lamps except the kitchen”, or “Siri, turn on the attic and corridor lights” [24].

Accessories, services, and features

A service is a feature of a device (accessory), for example, turning the light on or off. A device can have more than one accessory. For example, a garage door can allow to control the light and the door separately. A user also can create a group of services (corresponding to a group of devices) and control that group of devices (for example lights) independently of the rest of the devices in the same category (in this case lights). A characteristic is a controllable attribute of a device. For example, a characteristic of light can be the brightness, temperature, or color, and of a fan, it can be the speed of rotation [24].

Actions and scenes

One action corresponds to changing the characteristic of a service, for example, adjusting the fan speed or the light intensity of a lamp. Actions are initiated by users or through automation. A scene is a set of actions that control one or more services in one or more accessories. For example, the user can create a scenario called “Good morning” that turns on the lamps and turns on the coffee machine in the kitchen [24].

Automations

Automations perform actions based on a set of conditions such as specific times, days of the week, or when a device is activated (for example, a motion sensor) [24].

Zones

A zone represents an area of the home that contains multiple rooms, for example, the first floor or ground floor. The addition of zones is optional but has the advantage of controlling multiple accessories at once. Like using Siri to turn off all the lights on specific floor [24].

IoT Solution Providers

There are more than 100 [25] manufacturers around the world that are compatible and support Apple HomeKit, such as Lutron, Insteon, Philips, iHome, Nest, etc. All of these manufacturers must be registered with Apple's MFI (Made for iPhone / iPod / iPad) program, which gives access to technical specifications and all the necessary resources to create devices compatible with the Apple HomeKit.

UI and Screen

In figure 2.5 we have an UI example of the Apple Home smart home solution.

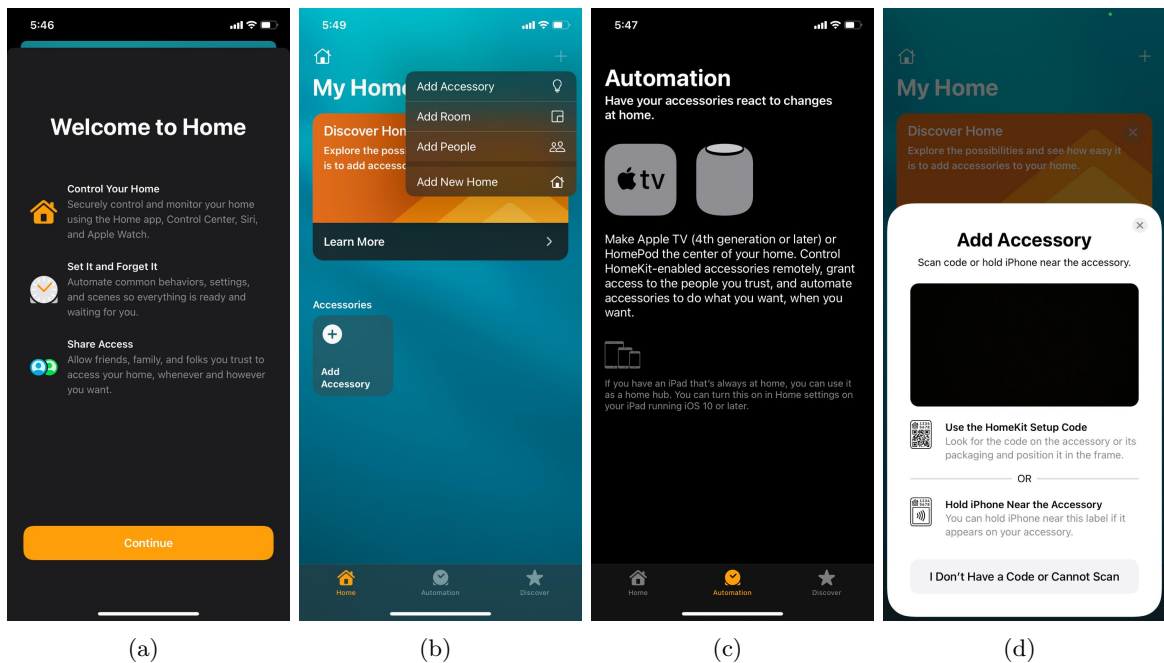


Figure 2.5: (a) Introduction (b) Home (c) Automation (d) Add device

2.4.3 TuyaSmart

Description

Tuya is an IoT services platform that enables products for consumers, brands, OEM manufacturers and retail chains [26]. The TuyaSmart and Smart Life App are both official Tuya App. The difference is that the Tuya Smart App uses the Tuya logo and elements, while the Smart Life App removes all Tuya logo and elements[27].The TuyaSmart or Smart Life allow the user to remotely control home appliances based on Android and iOS versions. It uses an originally created connection method called “Pegasus” that can detect new devices automatically achieving a one-click network configuration. It supports one-tap execution of combined automation smart scenes, and linkage between scenes and devices to enable interactions and interconnections between different products. It allows users to use third-party platforms such as Amazon Echo and Google Home to voice control devices [28, 29]. TuyaSmart or Smart Life provides easy control over smart devices, cloud storage for cameras, AI filtered notification, phone notification, message notification and, finally, the ability to know the status of devices at home in real-time. Tuya also offers another development solution, which is the Tuya OEM App. It is a solution that does not require additional development resources and is based on the Tuya official template. It provides some simple UI customization, personalized brand configuration and other information allowing brand-specific apps to be created within 3 days [30] and be ready to be released in the market. An example of that is the Lidl Home app.

Device communication

Wi-Fi Device

The device that use Wi-Fi module to connect the router, and interact with APP and cloud.

Quick Connection (EZ) Mode

Also known as the quick connection mode, the APP packs the network data packets into the designated area of the 802.11 data packets and sends them to the surrounding environment. The Wi-Fi module of the smart device is in the promiscuous model and monitors and captures all the packets in the network. According to the agreed protocol data format, it can parse out the network information packet sent by the APP.

Hotspot (AP) Mode

Also known as hotspot mode, the mobile phone connects the smart device’s hotspot. The two parties establish a Socket connection to exchange data through the agreed port.

Camera Scan Code Network Configuration

The camera device obtains the configuration data information by scanning the QR code on the APP.

Wired Network Configuration

Devices connected to the router via a wired network, such as ZigBee wired gateway, wired camera, etc.

Sub-device Configuration

Devices that interact with APP and cloud data through gateways, such as ZigBee sub-devices

BLE

Tuya Bluetooth has 3 technology lines.

- **SingleBLE** - Bluetooth single point device, one-to-one connection between Bluetooth device and phone.
- **TuyaMesh** - Mesh released by Tuya.
- **SigMesh** - Mesh released by SIG (Bluetooth Special Interest Group). In addition to the above three, there are some multi-protocol devices, such as Dual-mode Device with Wi-Fi and BLE capabilities. You can use both BLE and Wi-Fi capabilities.

Note: The references used in this all section was part of the Tuya glossary of terms [31]

Home Automation

Automation and Tap-to-run

The automation allows the App to automatically execute one or more tasks according to one or more conditions, such as the weather, device status, and time. For example, it is possible to automatically switch on the smart plug based on sunlight. The Tap-to-run is a predefined scenario that needs to be pressed in order to be activated as opposed to an automation.

Group devices

Link all smart devices in a group. For example, you can create a group for all appliances in the living room, a group for the garden, and a group for the kitchen. This way you can easily operate all devices that fall into the same group with one easy click.

Time switch

Depending on the type of device, you can set it to turn on or off at a specific time. For example, you can set the lamps to switch off automatically after 11:00 PM.

Timer

Is different from time switch, the timer sets up a specific period of operation. For example the user can be in the bed reading a book and set a timer for the lamps to switch off after 30mn, to make sure that in case he falls asleep there's no waste of energy.

Partners and IoT Solution Providers

Tuya connects the smart devices to various mainstream voice assistants in and outside China for users to control them with voice commands.

- **In China** - Xiaodu, Xiaowei, Tencent Jingle, DingDong, and Xiaoai.
- **Outside China** - Alexa Echo, Google Home, Yandex Alice and Siri. Tuya is also compatible with many of the same third-party services as the previous solutions as well as some others like DuerOS and DingDong.

UI and Screen

In figures 2.6 an 2.7 we have an UI example of the Tuya smart home solution.

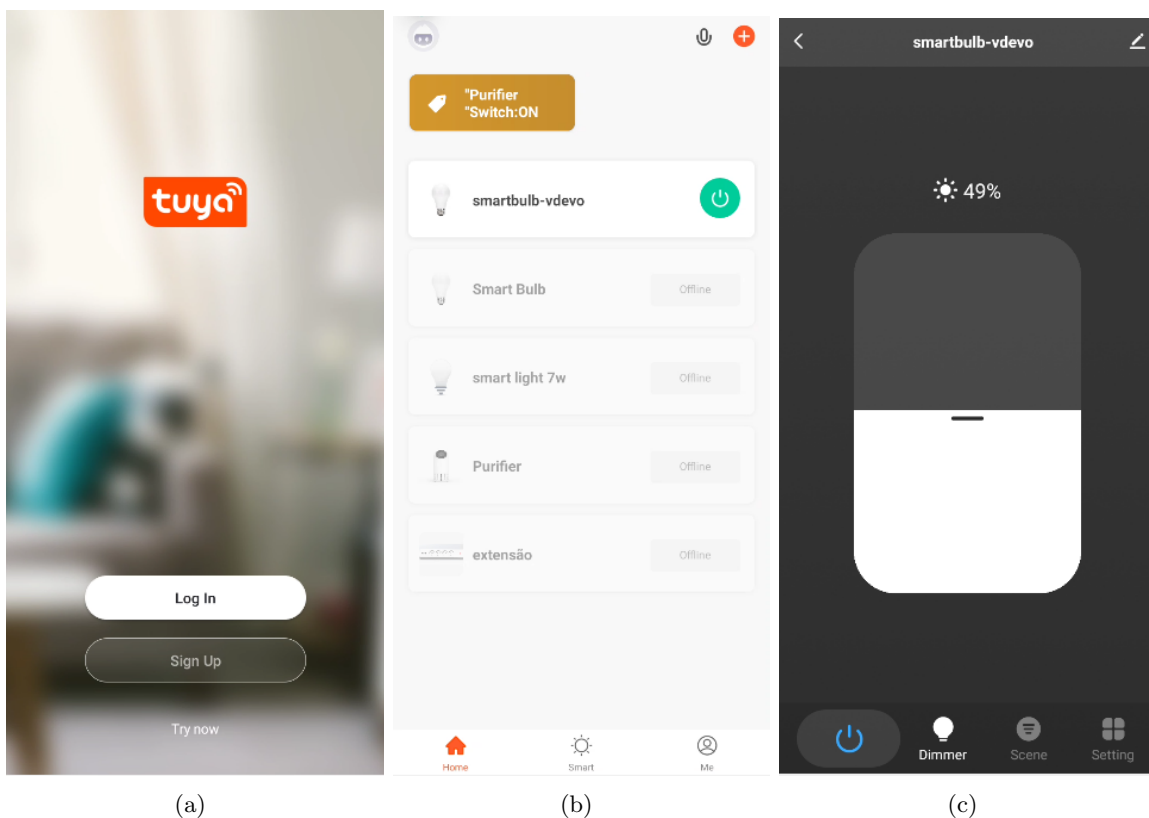


Figure 2.6: (a) Login (b) Home (c) Remote Control

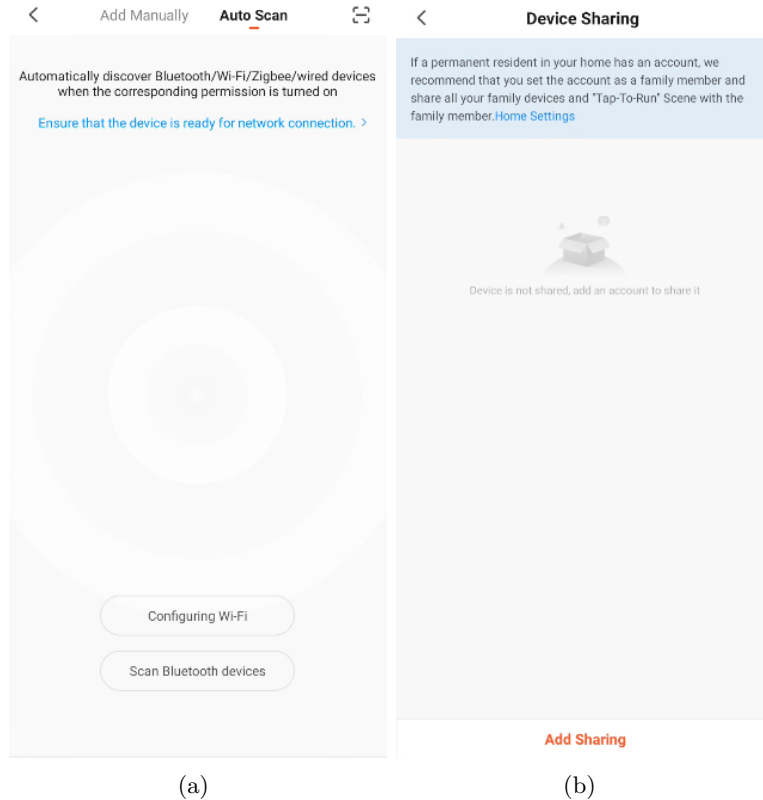


Figure 2.7: (a) Instant Connection (b) One tap to share

2.5 Which Solution is better

2.5.1 Compatible Devices

Device compatibility is one of the most important things to consider when building a new smart home. When it comes to that, SmartThings has the largest coverage of competitive IoT devices. Its usage of Zigbee and Z-wave protocols not only allows SmartThings to work with more devices, but also benefits from the fact that these wireless protocols use up less power than Wi-Fi, making it perfect for battery operated devices and sensors [32]. Alexa and Google Assistant are both compatible with a huge range of devices but there are still some devices that work with Alexa and not Google. Tuya is getting more recognition so more service providers are joining Tuya making even more devices available. Apple has a tight control over which devices it certifies to work with HomeKit, therefore there are fewer devices available for HomeKit [33].

2.5.2 Digital Assistants

When it comes to the smart home, digital assistants are used to control devices with the voice or to find information from the web, play music, and more. Both Alexa and Google Assistant are really capable assistants. There are plenty differences between the two but ultimately they achieve similar performance when it comes to daily tasks. Although Siri and

Bixby are getting better and better when compared to these two, they're still trailing a little.

	Alexa[34]	Google Assistant [35]	Siri	Bixby
Device compatibility	140,000+	50,000	exclusive to Apple	exclusive to Samsung
Automation	100,000+ skills	1,000,000+ actions	*1	*2
IoT brands	9,500+	10,000+	exclusive to apple	exclusive to Samsung
Languages	8	44	21	8

Table 2.1: Digital assistants features

**1- Siri doesn't have skills like Alexa or actions like Google Assistant, but on the flip-side it uses an app called Shortcuts which allows the user to quickly do everyday tasks, and with the most frequently used apps. Shortcuts are suggested right when you need them. Siri learns the app routines of the user and then suggests an easy way to perform common tasks.*

**2- Bixby learns the user patterns and recommends an automated routine of tasks so that it no longer has to spend time repeating the same task and chores.*

When discussing digital assistants its important to keep in mind that this technology is evolving quickly. This means that new services and features are constantly under development. Most of them perform similar tasks, but some excel when it comes to specific cases. In terms of device compatibility Alexa can be considered the best out of all the other assistants with over 140,000 compatible devices (from over 9,500 brands), and then Google Assistant with over 50,000 devices (from over 10,000) and lastly Siri and Bixby which only work with devices of their own brand. When it comes to automation, Google Assistant with its actions, and Alexa with its skills are leading the race, once more, providing the user with endless options of automations. Google Assistant takes the crown when it comes to assistant intelligence and language support [33].

2.5.3 Automation

Apple Homekit has an advantage when it comes to automation options but the other solutions aren't that far behind, they all offer similar capabilities. Some have different names but most of them offer the options of creating group devices, more then one home, scenes, routines and timers.

2.5.4 Extra features

Although a lot of smart home solutions are similar, each of them has something unique that distinguishes them from the others.

- Tuya offers the Tuya OEM App. It is a solution that does not require additional development resources. It is based on the tuya official template, provides some simple UI customization, personalized brand configuration and other information allowing brand-specific apps to be created within 3 days and ready to be released in the market. And that's a really good thing for developers.
- Apple Homekit has the Zone concept, which allow the user to give commands to all devices that are inserted in a zone, such as a garage, upstairs, attic, etc.
- Google Assistant can understand two commands at the same time, which is a step above Alexa's follow-up mode.
- Alexa's connection to Amazon offers the user an amazing shopping experience, it allows, throughout voice commands, to place, cancel, and rack orders.
- SmartApps of Samsung, allows the creation of really deep personalizations by experts, which can be in a totally different level of the scenes and automations of the other solutions.

2.5.5 Conclusion

When trying to set up a Smart Home ecosystem there are various factors to consider, from device compatibility, internet connectivity, available devices, among other things, but one of the most important factors is the user preferences and specific needs, because depending on these, one solution can be considered better than others. Ultimately, if the user already owns a few devices, the best solution will largely depend on that. There are some key benefits from each solution. For apple users with an iPhone, iPad, HomePod, Apple TV, or any other Apple smart products, HomeKit is simply going to be the most convenient solution, even if Siri lags behind a little and there aren't as many compatible smart devices to choose from. But HomeKit offers the most comprehensive smart home automation options and the best app interface [33]. For people who already own and use a lot of Android products, Google Assistant is likely a better bet. There's a good assortment of compatible devices to choose from, and the Google Assistant itself is the smartest out of the 4 voice assistants.

Chapter 3

Altice Home app

3.1 Introduction

Altice Home is a smart home solution that is being developed by Altice labs to be adapted to the needs of every household. It supports a unique ecosystem that allows thorough and effortless management of the house, interconnecting the various smart devices through a cloud solution and providing the user with a mobile App and a TV hub to manage the house in a smart and friendly way. Considering that the house needs permanent attention, this solution has integrated a virtual Buttlar (a Bot) that can be personalized and will assist the user in the management of the house and its goals. That way, he'll be always updated on the states of the devices, consumption, saving suggestions, or tips for a better house experience. The project is being developed by 3 teams, the UX team, responsible for the app design, the Cloud team, responsible for the APIs and Endpoints and the Developer team, responsible for the development of the app in Flutter.

3.2 Objectives

The key objective is the development of a mobile App using the Flutter mobile framework, that communicates with smart home Cloud and that allows controlling every supported device. The management is done using the App in an integrated way and with the possibility of:

- Devices and group management
- Device control
- Management of houses and divisions
- Notifications and pop-ups
- Interaction with MEO TV system
- Alexa and google skills interaction using BotSchool.

Future goals would be to invest in the mechanism of predictive behavior, energetic recommendations e other scenarios that take advantage of the information obtained from the devices, using machine learning to create artificial intelligence and make the houses truly smart.

** BOTSchool is a platform to create Virtual Assistants (BOTs) that can be used for the most diverse purposes of a company, from Support, to Purchasing Assistant, Self-Care, Management Assistant, among others. It uses Artificial Intelligence and Machine Learning techniques to provide a more intelligent and humanized conversation experience.*

3.3 Innovations aspects

Although the IoT world is quickly evolving in terms of smart home solutions, most of them can be too complex, not user-friendly, and lacking in terms of predictive behavior. Therefore, Altice Home pretends to develop a smart home concept that unifies the different ecosystems (electric, heating, security, health) of the house and allows the centralized management of every existing technology connected to the Mesh Wi-Fi network (developed by Altice Labs) using a simple and user-friendly approach.

3.4 Tools

The App is being developed using the Flutter framework which is a popular, multi-platform Google's UI toolkit powered by a single codebase, the Dart platform*, and that provides tooling and UI libraries to build UI experiences that run on iOS, Android, macOS, Windows, Linux, and the Web.

** Dart is a client-optimized language for developing fast apps on any platform. Its goal is to offer the most productive programming language for multi-platform development, paired with a flexible execution run-time platform for app frameworks*

3.5 Devices

Altice Home app is currently being developed and it's in the MVP (Minimum Valuable Product) phase. Therefore, the number of supported devices is limited. The solution supports a cheap starting kit, with the potential on helping to reach more buyers, which are also easy targets for up-selling on areas that Altice Labs has great know-how, such as Smart Mesh Wi-Fi, TV, Bots, and Health (SmartAL). The starter kit is a comprehensive set of low cost devices that are Smart Home compliant, easy to install and set up using the mobile App. For the starter kit, the following devices were considered:

- White and RGB lights (from Foshan and Yourlite vendors)
- Plugs (from Hangzhou Hongshi Eletrical Co)
- Cameras (from Nivian and MEARI vendors)

3.6 Architecture

The overall global architecture is represented in the figure 3.1. The solution comprehends several key components to support all the features that are envisioned in the project scope. The main components of the solution are:

- Cloud
- North Bound APIs to be used by the Mobile App
- Tuya SDKs
- Flutter Mobile App for Android and iOS

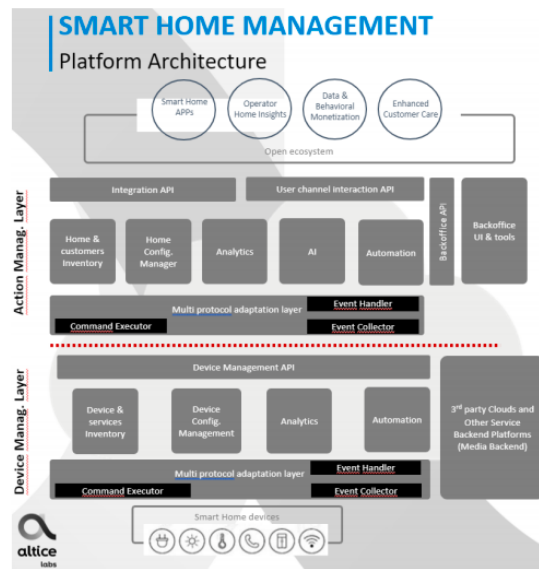


Figure 3.1: Global architecture

3.6.1 Cloud

The cloud is a service for securely connecting and managing devices. It enables backend code to run in response to events triggered by the App. The cloud is the central data hub of all connected devices. Some of its features include:

- Send commands to devices from the cloud
- Report state from devices to the cloud
- Detect when devices are offline
- Provide users access to manage their devices

3.6.2 APIs (Outsourced)

All the features available on the Application, namely, device management, scenario management, home, rooms, and others, are supported by specific REST APIs available on the Integration API module on the Cloud side. The APIs are HTTP/REST.

3.6.3 Tuya SDKs(Outsourced)

As stated in previous sections, the starter kit includes Tuya Wi-Fi devices only. Therefore, some Tuya SDKs are used to implement the functionalities needed on the App like device setup/onboarding and for the Smart Cameras control. Currently, Tuya devices can be added using several distinct modes as seen in previous sections (e.g., EZ mode, Ap mode, or Camera Scan code). In figure 3.2 we can see how the onboarding of devices is done and how the devices communicate with the App and the cloud. Basically the App communicates with the SDK, which then relays information to the cloud, ultimately ending up communicating with the device. New configurations are then passed again to the SDK, which allows the values to consequently change in the App side.

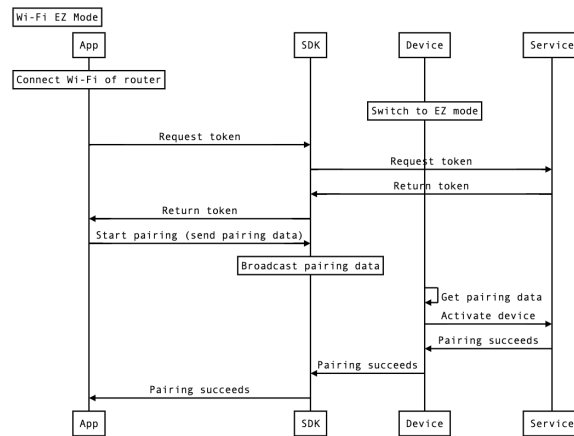


Figure 3.2: Diagram of process of pairing in Wi-Fi EZ mode by Tuya Inc.

3.6.4 Flutter Mobile App for Android and iOS

As previously stated, Flutter was the chosen framework used to develop the App. A detailed explanation this framework can be found in section 3.4. The screens implemented during this project using this platform were the following:

- Login Screen
- Room Screen
- User Account Screen
- Settings Screen

In the next chapter, we'll see a detailed explanation about each screen, regarding the design and implementation, but here we can see in figures 3.3 and 3.4 an UI example of the Altice Home App smart home solution.

The Home screen (figure 3.3 “a”) is the center of the Smart Home experience, it’s where everything connects and gets controlled. In this area we are presenting the scenarios if any exist, all the devices across multiple rooms. The devices are presented and grouped by room. If there no rooms, devices appear in a generic or default room. If we press any device or room we’ll be redirected to that specific device or room detail screen (figure 3.3 “d”) and pressing the “add” icon next to “My devices” redirects the user to a screen where he can set up new devices (figure 3.3 “c”). Also, as part of the Home screen, there is a left side menu with all the major areas of the application (figure 3.3 “b”). From there we can go to the Room and Scenarios areas where we can create new ones (figure 3.4 “a” and “b”).

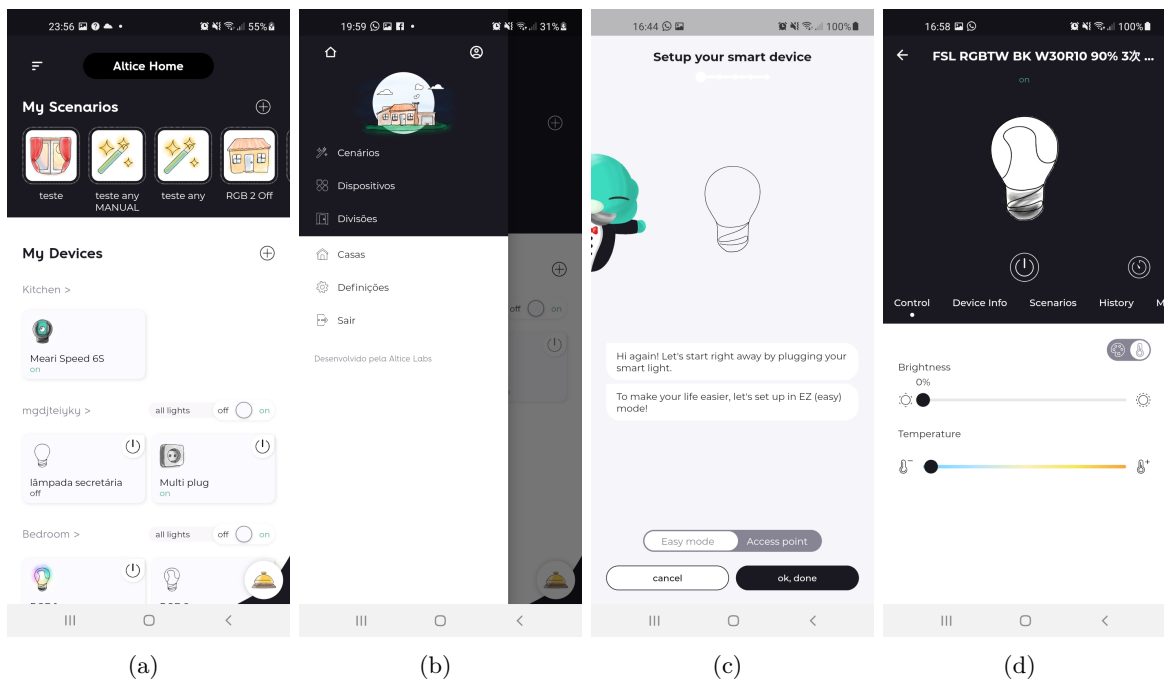


Figure 3.3: (a) Home (b) Home menu (c) Add device (d) Device info

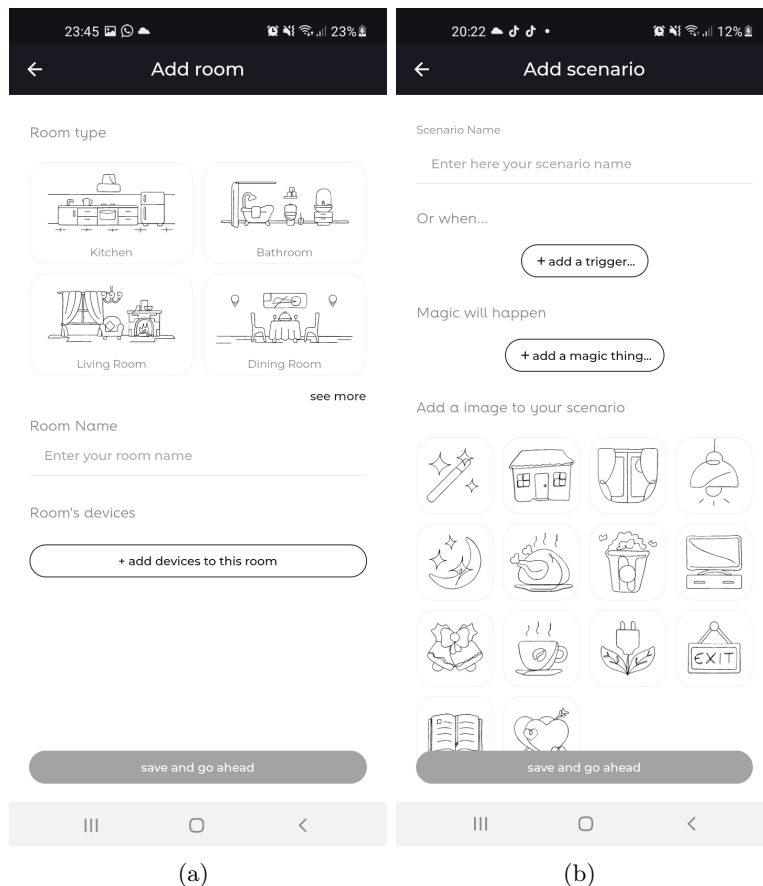


Figure 3.4: (a) Add room (b) Add scenario

Chapter 4

Design and Implementation

To help visualize what was implemented in the next sections the use case diagram in figure 4.1 provides a perspective of the system design from the end user's point-of-view, allowing to communicate the system's externally visible behavior in user terms [36].

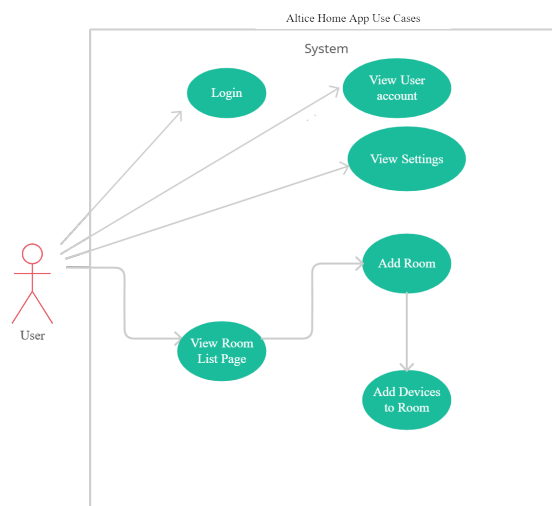


Figure 4.1: Use cases diagram

- **Login** - The user writes his account details, which are validated, and in case they are correct, he moves to the home page of the app otherwise receives an error message.
- **View User Account** - The user can verify his account information, like email, phone number, username, etc.
- **View Settings** - The user is presented with a list of configurations of the App he can later change, like the language, for example.
- **View Room List Page** - The user is presented with the list of every existing room, he's able to control all the lights of the rooms at the same time or individually. He is

also able to add another room and, when he does it, he's redirected to a page where he can select the new room details, like name, image or even add devices. When adding devices he's redirected to a page with a list of all the available devices in the account.

In the next sections, we'll have a description of the design and implementation of every screen implemented and in figures 4.2 and 4.3 there's presented an highlight of the most important functional and non-functional requirements.

Functional requirements - define if/then behaviors and include calculations, data input, and business process. Can also be thought of in terms of how the system responds to inputs. If the functional requirements are not met, the system will not work [37].

Non-functional requirements - do not affect the basic functionality of the system, even if they're not met the systems will still perform their basic purpose. They specify how the system should execute certain tasks [37].

	ID	Functional Requirements	ID	Non Functional Requirements
Login Screen	F1	<ul style="list-style-type: none"> Implement a login page where the user introduces his information and then the information is posted on the server for the authentication and in case the account already exists and the information is valid then the login is successful and the user is redirected to the home page, otherwise, it returns an error message. The eye icon should change when pressed, to simulate hide/show password 	NF1	<ul style="list-style-type: none"> show hint texts for username and password the password text should be obscured by default
Room Screen- Main	F2	<ul style="list-style-type: none"> Implement list of rooms that are associated to the user account and the list of devices that are associated in each room Implement 2 types of toggle buttons one inside of each room that has at least one light (allows to turn on or off every light present in the room) and another outside of the rooms (which allows to turn on or off all the lights in every room that has lights) Implement the button "add new room" that allows the user to navigate to the next screen where he can add a new room. 	NF2	<ul style="list-style-type: none"> The starter kit should not be included in the list The toggle button should only appear for rooms that have at list one light included In each room should only be displayed up to 4 devices, after that it should appear "4 device and the + more" sign
Room Screen- Add New Room	F3	<ul style="list-style-type: none"> The user able to create a new room. When the user selects a room, the colours change from black and white to coloured, to simulate room selected. The user can choose a name for the room and which device(s) he wants to add to the room. After pressing the "save and go ahead" button and gets redirected to the rooms main screen and the new room should appear in the list already. 	NF3	<ul style="list-style-type: none"> The user can only select one room at the same time When selecting the room the transitions of images should instantaneous

Figure 4.2: Functional and Non-Functional Requirements

Room Screen- Choose Devices	F4	<ul style="list-style-type: none"> The user gets a list with all the devices 	NF4	<ul style="list-style-type: none"> The user can choose one or more to add to the room.
User Account Screen	F5	<ul style="list-style-type: none"> Presents the user personal information 	NF5	<ul style="list-style-type: none"> All the user information should be displayed In the name avatar should appear the first letters of the first and last name
Settings Screen	F6	<ul style="list-style-type: none"> Presents a list of configurations the user can later change 	NF6	<ul style="list-style-type: none"> In the list each row needs to be equally spaced

Figure 4.3: Functional and Non-Functional Requirements

Note: Some screens are mainly UI based and therefore all the requirements are the ones defined in the specifications in the UXpin and are related to the components and to visual specs only, no logic included or limited one

4.1 Application Design

The application design was done using UXPin, which is a design tool ideal for interactive prototyping, design systems, and documentation. In this platform is where we have specified all of the functional and non-functional requirements. It supports variables, conditional interactions, expressions, interactive states, sitemaps, and data generators. The smart home has a Design system to better respond to different necessities. With a design system the project benefits in 3 major areas:

- **Efficiency** - Allow designers and developers to reuse components from a shared library, optimizing the work.
- **Consistency** - having a set of principles and rules to build the components, makes it easier to guarantee aligned and consistent experiences in different platforms and/or formats.
- **Scale** - by improving the efficiency and consistency it allows a faster development scale product. With the saved time the UX team can focus on other areas needed for the comprehension of the users and better success for the products.

Has indicated, the application design was done by the UX team. In the next part, the screens implemented under the scope of this dissertation will be described in detail.

4.2 Implementation

There are some concepts and good practices that were adopted to the majority of the screens, in order to reduce the complexity of the code and avoid boilerplate, they are called non-functional requirements and are the following :

- **The usage of cubit + freezed packages:** when developing an app, state management is an important factor to keep in mind, moreover mixing business logic and UI is not a good practice and makes the code hard to read. That is why cubit and freezed packages were used. Cubit is a lighter version of the Bloc package, with less boilerplate (section of code that is repeated in multiple places with little to no variation). This package is a well-known and established library when it comes to state management in Flutter. It promotes good practices such as immutability and it has one of the best ecosystems of supporting packages and documentation built around it. The freezed package supports sealed union via code generation, to represent mutually exclusive and immutable states with unidirectional data flow in the App. And guarantee that only valid states are allowed. Immutability is an important concept to keep in mind because it gives stricter control over the data making the code safer and more predictable. In other words, immutable objects allow to control the interface and data flow in a predictable manner, discovering the changes efficiently [38].

- **A text field factory class:** wherever there's a text box in the app we use a text field factory, and that can receive all of the following parameters, but depending on the subclass some of them might not be used, an example of this can be found at figure 4.4.

```

class ATextField extends StatefulWidget {
  final TextEditingController controller;
  final Function(String value) onChanged;
  final Function(String value) onSubmitted;
  final FocusNode focusNode;
  final String hintText;
  final TextStyle hintStyle;
  final bool obscureText;
  final VoidCallback onSuffixTapped;
  final VoidCallback onPrefixTapped;
  final String suffixSvgPath;
  final String prefixSvgPath;
  final String labelText;
  final TextInputType textInputType;
  final TextInputAction textInputAction;
  final bool enabled;
  final bool showBorder;
  final Color backgroundColor;
  final Color prefixColor;
  final Color suffixColor;
  final Brightness brightness;
}

```

Figure 4.4: Text field factory

- **AppStyle class-** A class that contains all the styles of text that are used throughout the app.
- **AppColor class-** A class that contains all the colors used in the app

Among these non-functional requirements we can also highlight the design specs and assets present in the UX platform such as:

- size and color of components
- typography
- screen size in terms of pixel (to guarantee that the layout remains the same regardless of the device being used)

4.3 Login Screen Design

This is a simple login page where the user introduces his information and then the information is posted on the server for the authentication. In case the account already exists and the information is valid then the login is successful otherwise it returns an error message.

In figure 4.5 we have the design of this screen, which aims to meet the requirements F1 and NF1 from the figure 4.2

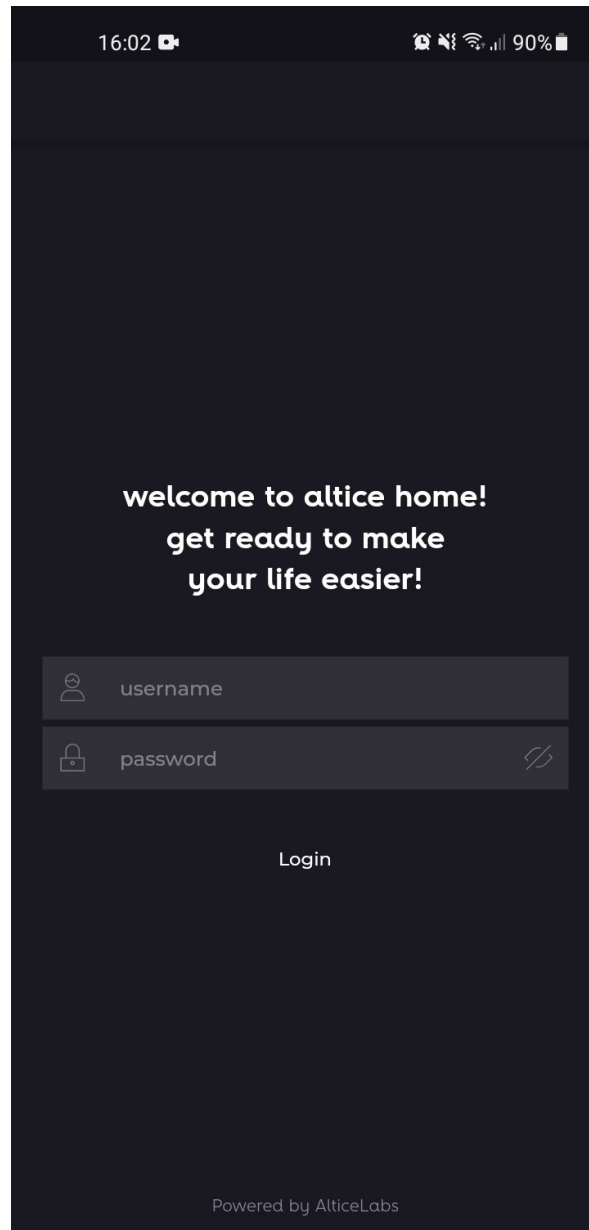


Figure 4.5: Login Screen

4.4 Login Screen Implementation

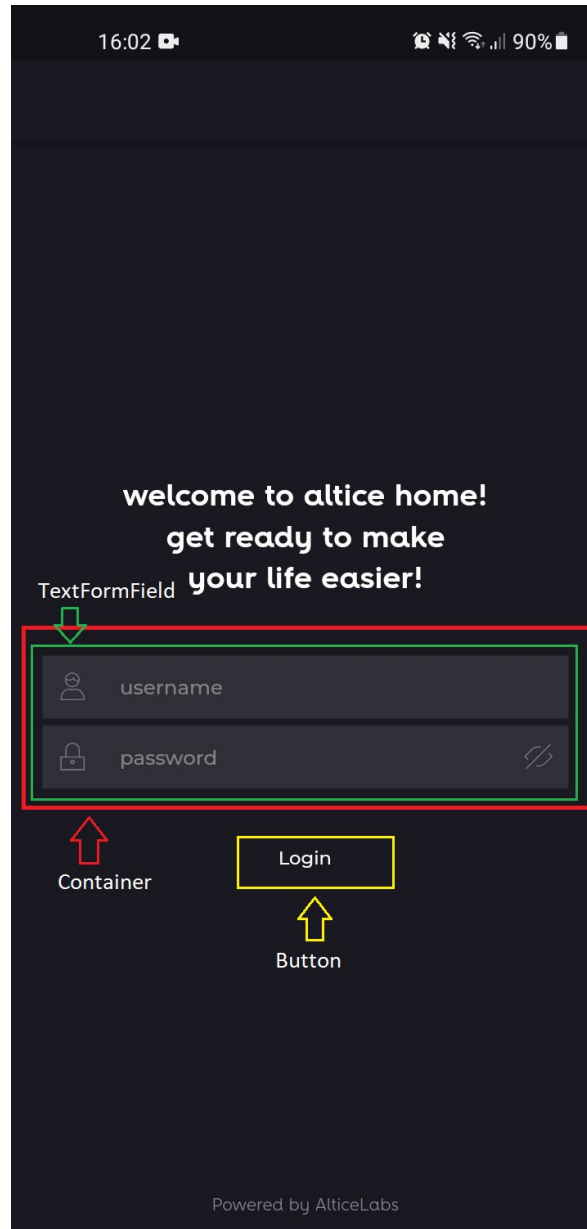


Figure 4.6: Login Screen with labels

In figure 4.6 we have an highlight of the most important components and widgets used that we'll be followed by a brief description.

KeyboardDismissOnTap- Removes the current focus and hides the keyboard when the user taps on the widget.

Scaffold- This is a single top-level container for a material app. It expands to fill the available space, usually, it means that it will occupy its entire window or device screen. On the body of

the scaffold we have a “BlocConsumer” (BlocConsumer<LoginPageCubit,LoginPageState>). The BlocConsumer exposes a “builder” and a “listener” to react to new states. It is analogous to a nested “BlocListener” and “BlocBuilder”, but reduces the amount of boilerplate needed. BlocConsumer is used in this case because it’s necessary to both rebuild the UI and execute other reactions to state changes in the “cubit”.

- The “listener” executes 4 different actions based on the 4 different states:
 - **Initial** - waits for the loading state and when loading is true it executes “Navigator.of(context).pop” to go back to the previous context.
 - **loading** - returns a “CircularProgressIndicator” while the authentication is being done.
 - **loginsucces** - in case of successful authentication, dismisses the CircularProgressIndicator and navigates to the homepage route.
 - **loginfailed** - returns a “snackbar” with the “invalid credentials” text
- The “builder” returns a widget based on the LoginPageCubit state and it’s where the UI is implemented.

```

ATextField.userNameLogin(
  controller: _usernameController,
), // ATextField.userNameLogin
const SizedBox(height: 5),
ATextField.passwordLogin(
  controller: _passwordController,
  hidePassword: state.hidePassword,
  onChangeObscureTapped: () =>
    BlocProvider.of<LoginPageCubit>(context).onShowHidePasswordTapped(),
), // ATextField.passwordLogin

```

Figure 4.7: Login Screen code excerpt

```

factory ATextField.userNameLogin({
  final TextEditingController controller,
}) {
  return ATextField(
    controller: controller,
    backgroundColor: AppColors.white.withOpacity(0.1),
    hintText: 'username',
    hintStyle: AppStyle.subtitle2.copyWith(color: AppColors.white.withOpacity(0.4)),
    prefixSvgPath: 'assets/svg/user.svg',
    prefixColor: AppColors.white.withOpacity(0.4),
    suffixColor: AppColors.white.withOpacity(0.4),
    brightness: Brightness.dark,
  );
}

factory ATextField.passwordLogin({
  final TextEditingController controller,
  final VoidCallback onChangeObscureTapped,
  final bool hidePassword = true,
}) {
  return ATextField(
    controller: controller,
    onSuffixTapped: onChangeObscureTapped,
    hintText: 'password',
    hintStyle: AppStyle.subtitle2.copyWith(color: AppColors.white.withOpacity(0.4)),
    obscureText: hidePassword,
    backgroundColor: AppColors.white.withOpacity(0.1),
    prefixSvgPath: 'assets/svg/lock.svg',
    suffixSvgPath: hidePassword ? 'assets/svg/invisible.svg' : 'assets/svg/visible.svg',
    prefixColor: AppColors.white.withOpacity(0.4),
    suffixColor: AppColors.white.withOpacity(0.4),
    brightness: Brightness.dark,
  );
}

```

Figure 4.8: Login Screen code excerpt

In this code excerpt from figures 4.7 and 4.8 we are invoking the text field factory and sending the controller parameter in the case of the username field and, in the case of the password, we’re also sending the controller value plus a boolean value to define the visibility of the password, and a call back function changes the value of this boolean value whenever the eye icon is pressed.

4.5 Room Screen Design

This screen is divided into 3 routes:

- Main screen
- Add new room
- Choose devices

4.5.1 Main Screen

In this screen we have the list of rooms that are associated to the user account and the list of devices that are associated in each room. We have two kinds of toggle buttons: one inside of each room that has at least one light (allows to turn on or off every light present in the room) and another outside of the rooms (which allows to turn on or off all the lights in every room that has lights). Finally, we have a the button "add new room" that allows the user to navigate to the next screen where he can add a new room.

In the figure 4.9 we have the design of this screen which aims to meet the requirements F2 and NF2 from the figure 4.2

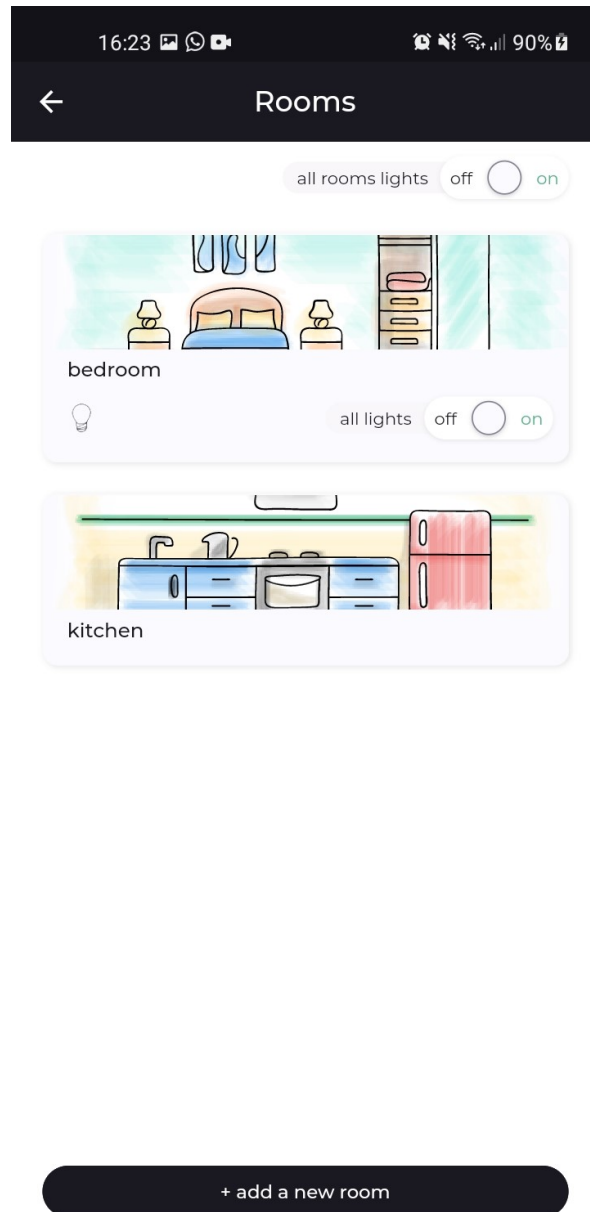


Figure 4.9: Room Main screen

4.5.2 Add New Room

In this screen the user can create a new room. When the user selects a room the colors change from black and white to colored, to simulate the room selected. After the room is selected the user can choose a name for the room and which device(s) he wants to add to the room. Finally, the user can press the “save and go ahead” button and get redirected to the rooms’ main screen. The new room should appear in the list.

In the figure 4.10 we have the design of this screen which aims to meet the requirements F3 and NF3 from the figure 4.2

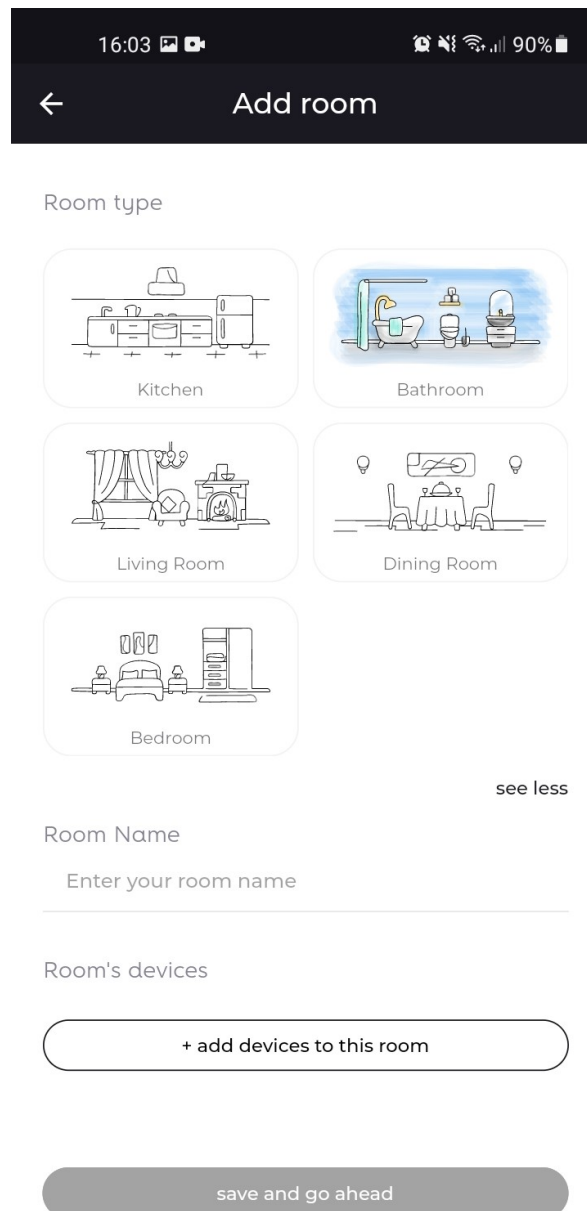


Figure 4.10: Add new room screen

4.5.3 Choose Devices

In this screen the user gets a list with all the devices. He can choose one or more to add to the room.

In the figure 4.11 we have the design of this screen which aims to meet the requirements F4 and NF4 from the figure 4.3



Figure 4.11: Choose Devices screen

4.6 Room Screen Implementation

4.6.1 Main Screen Implementation

In figure 4.12 we have an highlight of the most important components and widgets used that we'll be followed by a brief description. The header is implemented using the Scaffold appBar.

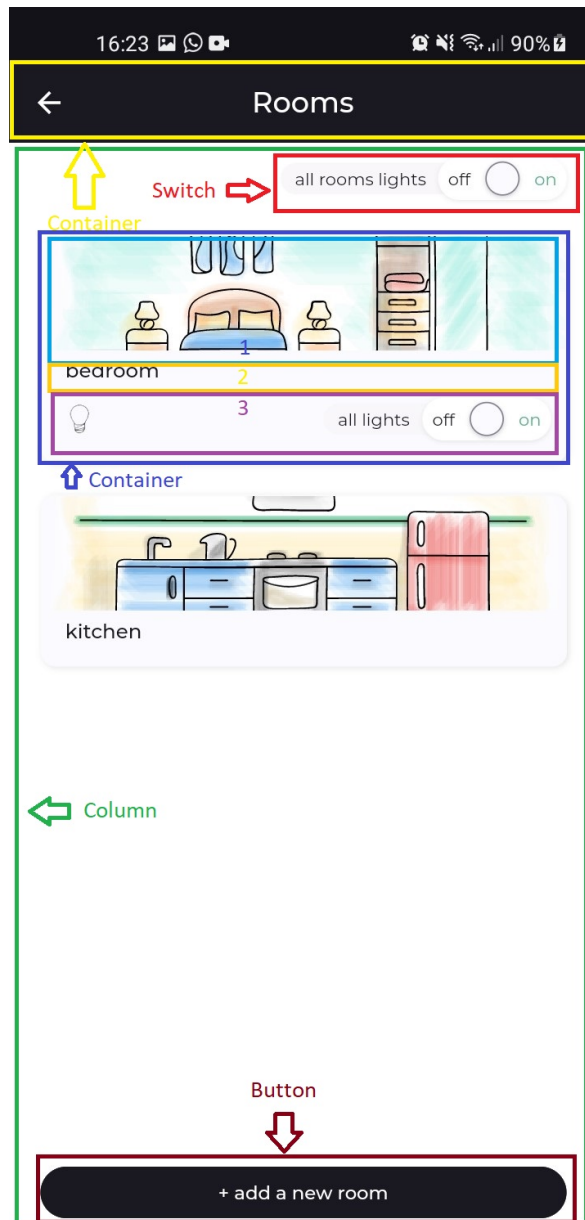


Figure 4.12: Main Screen with labels

In the excerpt of code on fig. 4.13 we have the next part of the UI. We have the column which contains the "all rooms switch" that allows to change the state of the lights when it is pressed through the callback function "cubit.onAllRoomsLightsSwitchChanged(true)", the cubit variable is equal to "BlocProvider.of<RoomsPageCubit>(context)". This switch is only visible if we have at least one smart light in any of the rooms. Next, we start by implementing the list of containers that implement all the rooms associated with the user account by calling the class "RoomCard()".

```
Widget _buildRoomsList(BuildContext context, List<RoomModel> rooms) {
  final cubit = BlocProvider.of<RoomsPageCubit>(context);

  return SingleChildScrollView(
    child: Column(
      children: [
        const SizedBox(height: 10),
        if (cubit.hasRoomsWithBulbDevices())
          Align(
            alignment: Alignment.centerRight,
            child: AllLightsSwitch(
              title: AppLocalizations.of(context).allRoomsLightsSwitch,
              onPressed: () => cubit.onAllRoomsLightsSwitchChanged(true),
              onOffTapped: () => cubit.onAllRoomsLightsSwitchChanged(false),
            ), // AllLightsSwitch
          ), // Align
        const SizedBox(height: 24),
        ...rooms
          .map(
            (room) => Padding(
              padding: const EdgeInsets.only(bottom: 22.0),
              child: RoomCard(
                roomModel: room,
                onPressed: () async {
                  await Navigator.of(context)
                    .pushNamed(RoomDetailPage.routeName, arguments: room);
                },
                onAllLightsSwitchChanged: (turnedOn) =>
                  cubit.onAllLightsSwitchChangedForRoom(room, turnedOn),
              ), // RoomCard
            ), // Padding
          )
          .toList(),
        SizedBox(height: MediaQuery.of(context).padding.bottom + 80),
      ],
    ), // Column
  ); // SingleChildScrollView
}
```

Figure 4.13: RoomPage excerpt

In the code excerpt of fig 4.14, we have a column with the 3 elements that belong to the room card container, firstly we have the room image, secondly the room name, and finally, we have a row with the room's available devices plus the switch that allows turning on/off all the lights that belong to that specific room. This switch is only visible if we have at least one smart light in the room.

```
child: Column(
  crossAxisAlignment: CrossAxisAlignment.stretch,
  children: [
    Container(
      height: 80,
      child: SvgPicture.asset(
        Utils.getSVGIllustrationPathForRoom(
          roomType: roomModel.type,
          colored: true,
        ),
      ),
      fit: BoxFit.cover,
    ), // SvgPicture.asset
  ], // Container
  const SizedBox(height: 5),
  Padding(
    padding: const EdgeInsets.symmetric(horizontal: 16.0),
    child: Text(
      roomModel.name,
      maxLines: 1,
      overflow: TextOverflow.ellipsis,
      style: AppStyle.subtitle2,
    ), // Text
  ), // Padding
  const SizedBox(height: 10),
  Padding(
    padding: const EdgeInsets.symmetric(horizontal: 10.0),
    child: Row(
      children: [
        Expanded(child: _buildDevicesList()),
        if (roomModel.hasLights())
          AlllightsSwitch(
            title: AppLocalizations.of(context).alllightsSwitch,
            onOnTapped: () => onAlllightsSwitchChanged(true),
            onOffTapped: () => onAlllightsSwitchChanged(false),
          ), // AlllightsSwitch
      ],
    ), // Row
  ), // Padding
),
```

Figure 4.14: RoomCard code excerpt

Finally, we have the room page cubit, where state management is done. In the code excerpt of fig 4.15 we have 2 functions. “loadRooms()” it’s a function that calls a cloud API that allows populating the “rooms” variable with all of the rooms associated with the user account and then in case of successful data retrieving the function “emitRooms(List <RoomModel >rooms)” emits the “loaded” state. The starter kit room type doesn’t appear on the room’s main screen page, only on the home page. In fig 4.17 we have an example of the constructor of the room model.

In the code excerpt of fig 4.16 we have 3 functions:

- void onAllLightsSwitchChangedForRoom(RoomModel room, bool turnedOn): It’s a function that updates the power capability state of all the lights in the room when the switch is pressed.
- void onAllRoomsLightsSwitchChanged(bool turnedOn): It’s a function similar to the one above, except it changes all the lights present in every room that has a light included.
- void onRoomCreated(RoomModel newRoom): Updates the list of rooms with a newly created room.

```
void _loadRooms() async {
  emit(RoomsPageState.loading(rooms: state.rooms));
  final rooms = await homesRepository.getRooms(homeId: Session.instance.currentHome.id);

  if (rooms != null) {
    _emitRooms(rooms);
  } else {
    emit(RoomsPageState.loadFailed(rooms: state.rooms));
  }
}

void _emitRooms(List<RoomModel> rooms) {
  emit(
    RoomsPageState.loaded(
      rooms: rooms.where((room) => room.type != RoomType.starterKit).toList(),
    ),
  );
}
```

Figure 4.15: RoomsPageCubit code excerpt

```
void onAllLightsSwitchChangedForRoom(RoomModel room, bool turnedOn) async {
  final result = await roomsRepository.updateDeviceTypeCapability(
    roomId: room.id,
    deviceType: DeviceType.bulb,
    capability: PowerCapability(value: turnedOn ? 'on' : 'off'),
  );

  if (!result) {
    _capabilitiesErrorSC.sink.add(true);
  }
}

void onAllRoomsLightsSwitchChanged(bool turnedOn) async {
  final result = await homesRepository.updateDeviceTypeCapability(
    homeId: Session.instance.currentHome.id,
    deviceType: DeviceType.bulb,
    capability: PowerCapability(value: turnedOn ? 'on' : 'off'),
  );

  if (!result) {
    _capabilitiesErrorSC.sink.add(true);
  }
}

void onRoomCreated(RoomModel newRoom) {
  emit(state.copyWith(rooms: state.rooms + [newRoom]));
}
```

Figure 4.16: RoomsPageCubit code excerpt

```

enum RoomType {
  starterKit,
  room,
  kitchen,
  bathroom,
  livingRoom,
  diningRoom,
  bedroom,
}

@Freezed
abstract class RoomModel implements _$RoomModel {
  const RoomModel._();
  const factory RoomModel({
    String id,
    String name,
    @Default(RoomType.room) RoomType type,
    @Default([]) List<DeviceModel> devices,
  }) = _RoomModel;

  bool hasLights() {
    if (type == RoomType.starterKit) {
      return false;
    }

    final device = devices.firstWhere((device) => device.isLight, orElse: () => null);
    return device != null;
  }

  factory RoomModel.fromJson(Map<String, dynamic> json) => _$RoomModelFromJson(json);
}

```

Figure 4.17: Room model constructor code excerpt

4.6.2 Add New Room Implementation

In figure 4.18 we have an highlight of the most important components and widgets used that we'll be followed by a brief description. The header is implemented using the Scaffold appBar.

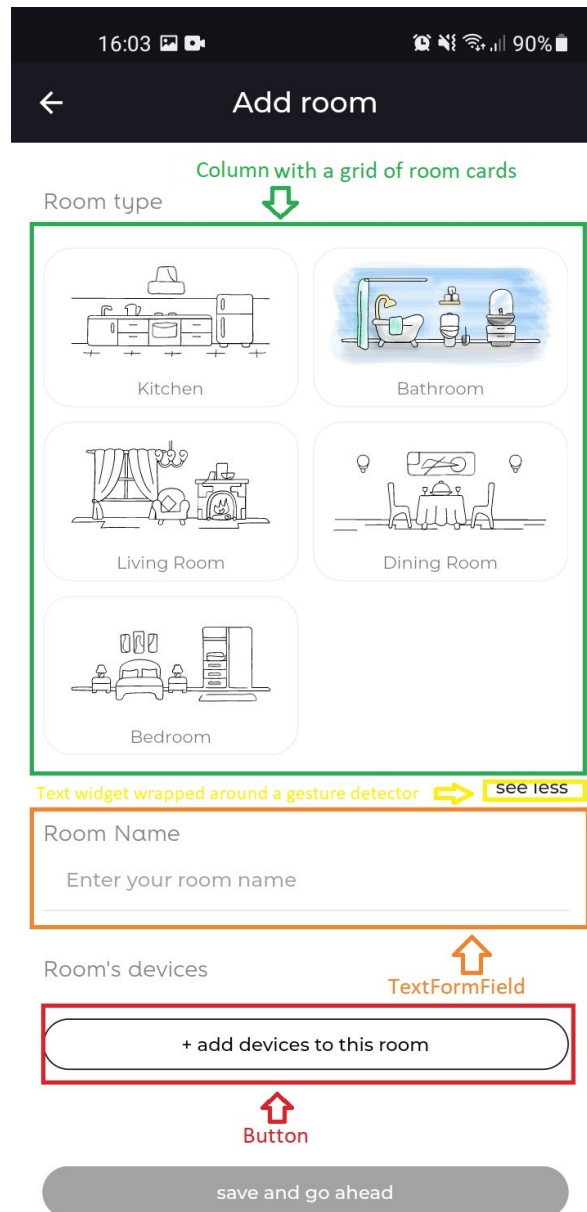


Figure 4.18: Add New Room Screen with labels

In the excerpt of code on fig. 4.19, we have the column which comprises most of the widgets from this screen. Firstly, we have the text widget “Room type”, secondly we have the function “buildRoomTypes()” which takes into consideration the app current context as well as all the room types that exist, the current value of the selected room type (the “RoomTypeSquareButton” class from “buildRoomTypes()” is simply for creating each room container with the help of the function “utils.getSVGIllustrationPathforRoom()” to change the image of each from colored to black and white depending on the room which is currently selected. We can see these excerpts in fig 4.21 and 4.22, and finally the boolean variable that controls the maximum length of rooms the user is allowed to see at once. In fig. 4.20 we have the implementation of this function, and we can see that it builds a grid in which the maximum length is defined by the number of different room types and the minimum is defined as 4. This value is controlled by the value of the boolean variable “expanded”, whose value is controlled by the text widget that is wrapped around by a gesture detector in fig 4.19.

```
Widget _buildBody(BuildContext context, AddRoomPageState state) {
  return SingleChildScrollView(
    child: Column(
      crossAxisAlignment: CrossAxisAlignment.stretch,
      children: [
        const SizedBox(height: 30),
        Text(
          AppLocalizations.of(context).roomType,
          style: AppStyle.subtitle1,
        ), // Text
        const SizedBox(height: 22),
        _buildRoomTypes(
          context,
          state.roomTypes,
          state.selectedRoomType,
          state.roomTypesExpanded,
        ),
        GestureDetector(
          behavior: HitTestBehavior.translucent,
          onTap: () => BlocProvider.of<AddRoomPageCubit>(context).onSeeMoreRoomTypesTapped(),
          child: SizedBox(
            height: 44,
            child: Align(
              alignment: Alignment.centerRight,
              child: Text(
                state.roomTypesExpanded
                  ? AppLocalizations.of(context).seeLess
                  : AppLocalizations.of(context).seeMore,
                style: AppStyle.button,
              ), // Text
            ),
          ),
        ),
      ],
    ),
  );
}
```

Figure 4.19: Add New Room Page excerpt

```
Widget _buildRoomTypes(
  BuildContext context,
  List<RoomType> roomTypes,
  RoomType selectedRoomType,
  bool expanded,
) {
  return GridView.builder(
    padding: EdgeInsets.zero,
    shrinkWrap: true,
    physics: NeverScrollableScrollPhysics(),
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
      childAspectRatio: 16 / 10,
      crossAxisCount: 2,
      crossAxisSpacing: 10,
      mainAxisSpacing: 10,
    ), // SliverGridDelegateWithFixedCrossAxisCount
    itemCount: expanded ? roomTypes.length : min(4, roomTypes.length),
    itemBuilder: (context, index) {
      final roomType = roomTypes[index];

      return RoomTypeSquareButton(
        roomType: roomType,
        selected: selectedRoomType == roomType,
        onTap: () => BlocProvider.of<AddRoomPageCubit>(context).onRoomTypeSelected(roomType),
      ); // RoomTypeSquareButton
    },
  ); // GridView.builder
}
```

Figure 4.20: Add New Room Page excerpt

```

@override
Widget build(BuildContext context) {
  return GestureDetector(
    behavior: HitTestBehavior.translucent,
    onTap: onTap,
    child: Container(
      decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(20),
        border: Border.all(
          color: AppColors.waterloo.withOpacity(0.1),
        ), // Border.all
      ), // BoxDecoration
      child: Column(
        children: [
          Expanded(
            child: SvgPicture.asset(
              Utils.getSVGIllustrationPathforRoom(
                roomType: roomType,
                colored: selected,
              ),
            ), // SvgPicture.asset
          ), // Expanded
          Text(
            Utils.getRoomNameForType(roomType: roomType, context: context),
            style: AppStyle.body2.copyWith(
              color: AppColors.bastille.withOpacity(0.6),
            ),
          ),
        ],
      ),
    ),
  );
}

```

Figure 4.21: RoomTypeSquareButton class code excerpt

```

static String getSVGIllustrationPathforRoom({
  @required RoomType roomType,
  @required bool colored,
}) {
  String basePath = 'assets/svg/rooms/illustrations/';

  switch (roomType) {
    case RoomType.livingRoom:
      return basePath + (colored ? 'living-room-Sc.svg' : 'living-room-S.svg');
    case RoomType.bedroom:
      return basePath + (colored ? 'bedroom-Sc.svg' : 'bedroom-S.svg');
    case RoomType.kitchen:
      return basePath + (colored ? 'kitchen-Sc.svg' : 'kitchen-S.svg');
    case RoomType.starterKit:
      return basePath + (colored ? 'kitchen-Sc.svg' : 'kitchen-S.svg');
    case RoomType.room:
      return basePath + (colored ? 'room-default-Sc.svg' : 'room-default-S.svg');
    case RoomType.bathroom:
      return basePath + (colored ? 'bathroom-Sc.svg' : 'bathroom-S.svg');
    case RoomType.diningRoom:
      return basePath + (colored ? 'dining-room-Sc.svg' : 'dining-room-S.svg');
  }

  return basePath + (colored ? 'kitchen-Sc.svg' : 'room-default-S.svg');
}

```

Figure 4.22: utils.getSVGIllustrationPathforRoom code excerpt

In the code excerpt of fig 4.23 and 4.24, we have the “+ add devices to this room” button that redirects to the “choose devices page” when pressed and the “save and go ahead button” that creates a new room with all the information that was selected above.

```
Center(  
  child: SecondaryButton(  
    title: AppLocalizations.of(context).addDeviceToRoom,  
    onTap: () async {  
      final selectedDevices = await Navigator.of(context).pushNamed(  
        SelectDevicesPage.routeName,  
        arguments: SelectDevicesPageArguments(  
          devicesAdded: state.devices,  
        ), // SelectDevicesPageArguments  
      );  
    }  
  );  
);
```

Figure 4.23: Add New Room Page excerpt

```
child: KeyboardVisibilityBuilder(  
  builder: (context, visible) {  
    return visible  
      ? Container()  
      : PrimaryButton(  
        title: AppLocalizations.of(context).saveAndGoAhead,  
        enabled:  
          state.selectedRoomType != null && state.roomName.isNotEmpty,  
        onTap: () async {  
          final roomModel =  
            await BlocProvider.of<AddRoomPageCubit>(context).onSave();  
  
          if (roomModel != null) {  
            Navigator.of(context).pop(roomModel);  
          } else {  
            Utils.showToast(  
              scaffoldState: Scaffold.of(context),  
              title: AppLocalizations.of(context).somethingWentWrong,  
              toastType: ToastType.error,  
            );  
          }  
        }  
      );  
  }); // PrimaryButton
```

Figure 4.24: Add New Room Page excerpt

Finally, we have the “add new room page cubit”, where state management is done. In the code excerpt of fig 4.25 we have 3 important functions:

- “onRoomTypeSelected()” it’s a function that updates the value of the selected room type;
- “onRoomNameChanged()” it’s a function that updates the name of the selected room
- “onSave()” it’s a function that creates the new room, with all the new pieces of information, room type, room name, and devices.

```
void onRoomTypeSelected(RoomType roomType) {
  emit(state.copyWith(selectedRoomType: roomType));
}

void onRoomNameChanged(String name) {
  emit(state.copyWith(roomName: name));
}

Future<RoomModel> onSave() async {
  return await roomsRepository.createRoom(
    RoomModel(
      type: state.selectedRoomType,
      name: state.roomName,
      devices: state.devices,
    ),
  );
}
```

Figure 4.25: Add New Room Page Cubit code excerpt

4.6.3 Choose Devices Implementation

The screen of figure 4.26 is mainly UI and there's no logic implemented. Therefore, there is no need to present an excerpt of code because the widget implementation used is very similar to several ones described before.

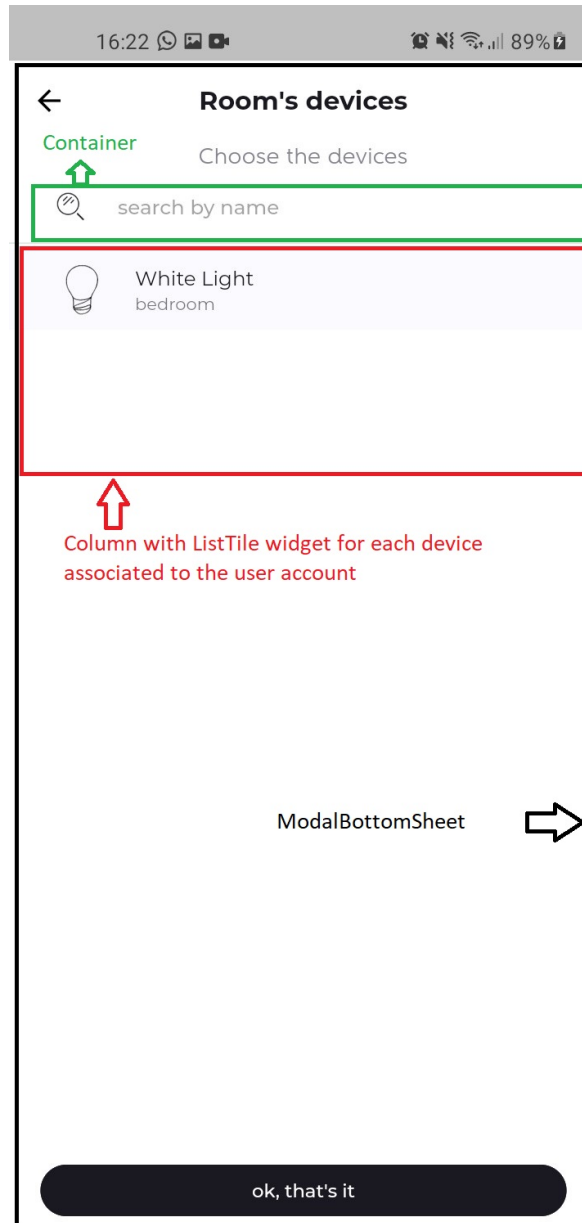


Figure 4.26: Choose Devices screen with labels

Note: This was the first set of screens developed by me as a Flutter developer. Hence, there were a lot of aspects where I lacked knowledge and experience. The code above is an upgraded version of the code developed by me that suffered a lot of changes from my colleague who performed the code review. In the next section, I'll show the first version of my code and conclude what were the best

practices adopted by my colleague and how they contributed to my knowledge and evolution for the other screens.

4.7 Room Screen 1st Implementation

4.7.1 Main Screen 1st Implementation

In the upgraded implementation what was achieved by the function “buildroom” and the class “RoomCard” from fig 4.13 and 4.14 respectively, was implemented in the same class by the functions “buildRooms” and “buildRoom”. Analyzing both of the code excerpts from figures 4.27 to 4.30 we can notice a number of differences. In the first implementation we can see there’s a lot of mix of logic and UI, and the code is not very clear, while in the upgraded one, everything is well organized and divided by classes.

```
Widget _buildLoaded(BuildContext context, List<RoomModel> roomModel) {
  return Column(
    children: [
      _buildTop(context, roomModel),
      Expanded(
        child: SingleChildScrollView(
          child: Column(
            children: [
              const SizedBox(height: 30),
              Padding(
                padding: const EdgeInsets.symmetric(horizontal: 20.0),
                child: Row(
                  mainAxisAlignment: MainAxisAlignment.spaceBetween,
                  children: [
                    Spacer(),
                    AllLightSwitch(
                      onTap: () => BlocProvider.of<RoomPageCubit>(context)
                        .onAllLightRoomOnTapped(roomModel.last),
                      onOffTapped: () => BlocProvider.of<RoomPageCubit>(context)
                        .onAllLightRoomOffTapped(roomModel.last), // AllLightSwitch
                    ),
                  ],
                ), // Row
              ), // Padding
              const SizedBox(height: 20),
              _buildRooms(context, roomModel),
            ],
          ), // Column
        ), // SingleChildScrollView
      ), // Expanded
      _buildAddButton(context),
    ],
  );
}
```

Figure 4.27: RoomPage excerpt

```
Widget _buildRooms(BuildContext context, List<RoomModel> rooms) {
  return Column(
    children: rooms
      .map((room) => Padding(
        padding: const EdgeInsets.only(bottom: 30.0),
        child: _buildRoom(context, room),
      )) // Padding
      .toList(),
  ); // Column
}
```

Figure 4.28: RoomPage excerpt

```

Widget buildRoom(BuildContext context, RoomModel room) {
  if (room.devices == null || room.type == RoomType.starterKit) {
    return Center(
      child: Text('No devices for this room'),
    ); // Center
  }

  String svgPath;
  switch (room.type) {
    case RoomType.kitchen:
      svgPath = 'assets/svg/Kitchen-Off-M.svg';
      break;

    case RoomType.bedroom:
      svgPath = 'assets/svg/Bedroom-Off-M.svg';
      break;

    default:
      svgPath = 'assets/svg/Outside-Off-M.svg';
      break;
  }

  return Container(
    height: 180,
    decoration: BoxDecoration(
      color: AppColors.secondary,
      borderRadius: BorderRadius.only(
        topLeft: Radius.circular(10),
        topRight: Radius.circular(10),
        bottomLeft: Radius.circular(10),
        bottomRight: Radius.circular(10),
      ), // BorderRadius.only
  ),
);

```

Figure 4.29: RoomPage excerpt

```

Container(
  padding: EdgeInsets.only(left: 10),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      Text('${room.name}', style: AppStyle.body1.copyWith(fontWeight: FontWeight.bold)),
      Spacer(),
      if (room.hasLights() && room.type != RoomType.starterKit)
        Padding(
          padding: const EdgeInsets.only(right: 10.0),
          child: AllLightsSwitch(
            onPressed: () =>
              BlocProvider.of<RoomPageCubit>(context).onAllLightRoomOnTapped(room),
            onOffTapped: () =>
              BlocProvider.of<RoomPageCubit>(context).onAllLightRoomOffTapped(room),
            ), // AllLightsSwitch
          ), // Padding
    ],
  ), // Row
), // Container
_buildDevices(room.devices),

```

Figure 4.30: RoomPage excerpt

In the upgraded version, in the cubit (figures 4.31 and 4.32), we focus on loading the rooms, and changing the status of the switches, and creating new rooms. In the initial version, we also load the rooms and change the value of the switch as we update the current state of the room. In the initial version, the cloud APIs weren't available yet, so I was using a dummy provider for obtaining all the house information.

```

void _updateDeviceStatus(DeviceModel device, DeviceOperStatus status) {
    final room = state.roomModel.firstWhere(
        (room) => room.devices.firstWhere((d) => d.id == device.id, orElse: () => null) != null);
    final roomIndex = state.roomModel.indexOf(room);

    final updatedDevicesList = List<DeviceModel>.from(room.devices);
    final updatedDevice = device.copyWith(operStatus: status);
    final deviceIndex = room.devices.indexOf(device);
    updatedDevicesList.removeAt(deviceIndex);
    updatedDevicesList.insert(deviceIndex, updatedDevice);

    final updatedRoom = room.copyWith(devices: updatedDevicesList);
    final updatedRooms = List<RoomModel>.from(state.roomModel);
    updatedRooms.removeAt(roomIndex);
    updatedRooms.insert(roomIndex, updatedRoom);

    emit(RoomPageState.loadedRoom(roomModel: updatedRooms));
}

void onAlllightRoomOnTapped(RoomModel room) {
    for (final device in room.devices) {
        if (device.isLight && device.operStatus == DeviceOperStatus.off) {
            _updateDeviceStatus(device, DeviceOperStatus.on);
        }
    }
}

void onAlllightRoomOffTapped(RoomModel room) {
    for (final device in room.devices) {
        if (device.isLight && device.operStatus == DeviceOperStatus.on) {
            _updateDeviceStatus(device, DeviceOperStatus.off);
        }
    }
}

```

Figure 4.31: RoomsPageCubit code excerpt

```

void _loadRoom() async {
    try {
        final room = await roomsRepository.getRooms();
        emit(RoomPageState.loadedRoom(roomModel: room));
    } catch (e) {}
}

void onDeviceButtonTapped(DeviceModel device) {
    switch (device.operStatus) {
        case DeviceOperStatus.on:
            case DeviceOperStatus.off:
                _updateDeviceOnOff(device);
                break;
            case DeviceOperStatus.disconnected:
                print("Device Disconnected, do nothing");
                break;
            case DeviceOperStatus.notInstalled:
                break;
    }
}

void _updateDeviceOnOff(DeviceModel device) {
    _updateDeviceStatus(device,
        device.operStatus == DeviceOperStatus.on ? DeviceOperStatus.off : DeviceOperStatus.on);
}

```

Figure 4.32: RoomsPageCubit code excerpt

4.7.2 Add New Room 1st Implementation

The first part of the implementation is similar to the upgraded implementation. We start encountering the first differences in the "builRoomTypes", "RoomTypeSquareButton" and "utils.getSVGIllustrationPathforRoom" (fig 4.19, 4.21 and 4.22 from the upgraded version). The upgraded version uses the cubit to update the state of the variable corresponding to the currently selected room, has every significant step in its class, and is simply organized. In the first version, there's logic (for and if cycles) mixed with UI and setState (as we can see in the excerpt of the codes from figures 4.33 to 4.35), which ends up breaking clean code principles. This is a big problem when we have an app with a lot of screens because it scatters the state all over the place. "setState" should only be used when the state only exists within a widget and no other part of the app needs access to this state.

```
Widget build(BuildContext context) {
  return GridView.builder(
    shrinkWrap: true,
    padding: EdgeInsets.zero,
    physics: NeverScrollableScrollPhysics(),
    gridDelegate: SliverGridDelegateWithFixedCrossAxisCount(
      crossAxisCount: 4,
      childAspectRatio: 1,
      crossAxisSpacing: 12,
      mainAxisSpacing: 12,
    ), // SliverGridDelegateWithFixedCrossAxisCount
    itemCount: 8,
    itemBuilder: (context, index) {
      if (!init) {
        for (int i = 0; i < 8; i++) {
          tapped[i] = false;
        }
        init = true;
      }
      return GestureDetector(
        onTap: () {
          auxIndex = index;
          setState(() {
            tapped[auxIndex] = !tapped[auxIndex];
            changed = true;
          });
        },
        child: Container(
```

Figure 4.33: Add New Room Page code excerpt

```
child: Container(
  decoration: BoxDecoration(
    color: AppColors.white,
    borderRadius: BorderRadius.only(
      topLeft: Radius.circular(10),
      topRight: Radius.circular(10),
      bottomLeft: Radius.circular(10),
      bottomRight: Radius.circular(10),
    ), // BorderRadius.only
    boxShadow: [
      BoxShadow(
        color: Colors.black.withOpacity(0.1),
        spreadRadius: 0,
        blurRadius: 2,
        offset: Offset(1, 1),
      ), // BoxShadow
    ],
  ), // BoxDecoration
  child: SvgPicture.asset(RoomUtils.getSVGPathForRoomImage(
    index: index, tapped: tapped, auxIndex: auxIndex, change: changed)),
)); // Container // GestureDetector
}); // GridView.builder
```

Figure 4.34: Add New Room Page code excerpt

```

class RoomUtils {
    static String getSVGPathforRoomImage({
        @required int index,
        @required List<bool> tapped,
        @required int auxIndex,
        @required bool change,
    }) {
        for (int i = 0; i < tapped.length; i++) {
            switch (index) {
                //goes through every index of the list
                case 0:
                    if (index == auxIndex && change) {
                        change = false;
                        return (tapped[index])
                            ? 'assets/svg/ImHome_Line-On.svg'
                            : 'assets/svg/ImHome_Line-Off.svg';
                    } else {
                        return 'assets/svg/ImHome_Line-Off.svg';
                    }
                    break;
                case 1:
                    if (index == auxIndex && change) {
                        change = false;
                        return (tapped[index])
                            ? 'assets/svg/GoodMorning_Line-On.svg'
                            : 'assets/svg/GoodMorning_Line-Off.svg';
                    }
            }
        }
    }
}

```

Figure 4.35: Add New Room Page code excerpt

In the excerpt of code on figure 4.36 we have the widget “buildRooms” which a column where each element (the room) is implemented by the widget “buildRoom” from figures 4.37 and 4.38(4.19 from upgraded version), we have the column which comprises most of the widgets from this screen. The code excerpts from figures 4.33 to 4.35 is where we simulate the selection of rooms.

```
Widget buildRooms(BuildContext context, List<RoomModel> rooms) {
  return Column(
    children: rooms
      .map((room) => Padding(
        padding: const EdgeInsets.only(bottom: 30.0),
        child: buildRoom(context, room),
      )) // Padding
      .toList(),
  ); // Column
}
```

Figure 4.36: Add New Room Page excerpt

```
Widget buildRoom(BuildContext context, RoomModel room) {
  if (room.devices == null || room.type == RoomType.starterKit) {
    return Center(
      child: Text('No devices for this room'),
    ); // Center
  }

  String svgPath;
  switch (room.type) {
    case RoomType.kitchen:
      svgPath = 'assets/svg/Kitchen-Off-M.svg';
      break;

    case RoomType.bedroom:
      svgPath = 'assets/svg/Bedroom-Off-M.svg';
      break;

    default:
      svgPath = 'assets/svg/Outside-Off-M.svg';
      break;
  }

  return Container(
    height: 180,
    decoration: BoxDecoration(
      color: AppColors.secondary,
      borderRadius: BorderRadius.only(
        topLeft: Radius.circular(10),
        topRight: Radius.circular(10),
        bottomLeft: Radius.circular(10),
        bottomRight: Radius.circular(10),
      ), // BorderRadius.only
  );
```

Figure 4.37: Add New Room Page excerpt

```

Container(
  padding: EdgeInsets.only(left: 10),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      Text('${room.name}', style: AppStyle.body1.copyWith(fontWeight: FontWeight.bold)),
      Spacer(),
      if (room.hasLights() && room.type != RoomType.starterKit)
        Padding(
          padding: const EdgeInsets.only(right: 10.0),
          child: AllLightsSwitch(
            onPressed: () =>
              BlocProvider.of<RoomPageCubit>(context).onAllLightRoomOnTapped(room),
            onOffTapped: () =>
              BlocProvider.of<RoomPageCubit>(context).onAllLightRoomOffTapped(room),
            ), // AlllightsSwitch
          ), // Padding
    ],
  ), // Row
), // Container
buildDevices(room.devices),

```

Figure 4.38: Add New Room Page excerpt

In the code excerpt of fig 4.39 and 4.40, we have the “+ add devices to this room” button that redirects to the “choose devices page” when pressed and the “save and go ahead button” that creates a new room with all the information that was selected above.

```
Center(  
  child: SecondaryButton(  
    title: AppLocalizations.of(context).addDeviceToRoom,  
    onTap: () async {  
      final selectedDevices = await Navigator.of(context).pushNamed(  
        SelectDevicesPage.routeName,  
        arguments: SelectDevicesPageArguments(  
          devicesAdded: state.devices,  
        ), // SelectDevicesPageArguments  
      );  
    }  
  );  
);
```

Figure 4.39: Add New Room Page excerpt

```
child: KeyboardVisibilityBuilder(  
  builder: (context, visible) {  
    return visible  
      ? Container()  
      : PrimaryButton(  
        title: AppLocalizations.of(context).saveAndGoAhead,  
        enabled:  
          state.selectedRoomType != null && state.roomName.isNotEmpty,  
        onTap: () async {  
          final roomModel =  
            await BlocProvider.of<AddRoomPageCubit>(context).onSave();  
  
          if (roomModel != null) {  
            Navigator.of(context).pop(roomModel);  
          } else {  
            Utils.showToast(  
              scaffoldState: Scaffold.of(context),  
              title: AppLocalizations.of(context).somethingWentWrong,  
              toastType: ToastType.error,  
            );  
          }  
        }  
      );  
  }); // PrimaryButton
```

Figure 4.40: Add New Room Page excerpt

Finally, we have the “add new room page cubit”, where state management is done. In the code excerpt of fig 4.41 we have 3 important functions:

- ”onRoomTypeSelected()” it’s a function that updates the value of the selected room type;
- ”onRoomNameChanged()” it’s a function that updates the name of the selected room
- ”onSave()” it’s a function that creates the new room, with all the new pieces of information, room type, room name, and devices.

```
void onRoomTypeSelected(RoomType roomType) {
    emit(state.copyWith(selectedRoomType: roomType));
}

void onRoomNameChanged(String name) {
    emit(state.copyWith(roomName: name));
}

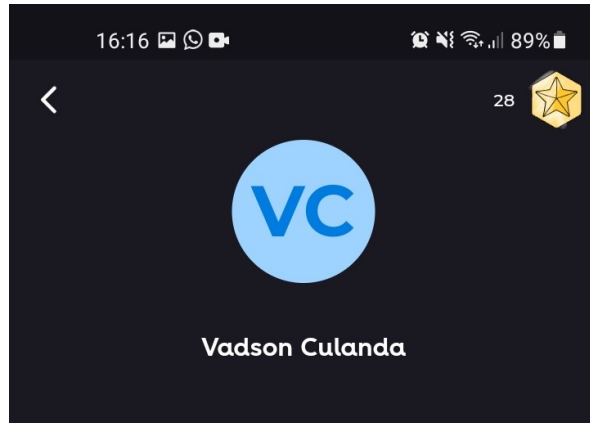
Future<RoomModel> onSave() async {
    return await roomsRepository.createRoom(
        RoomModel(
            type: state.selectedRoomType,
            name: state.roomName,
            devices: state.devices,
        ),
    );
}
```

Figure 4.41: Add New Room Page Cubit code excerpt

4.8 User Account Screen Design

This is a simple read only page the contains the user basic information.

In the figure 4.42 we have the design of this screen which aims to meet the requirements F5 and NF5 from the figure 4.3



User name

Vadson Culanda

Email

vadson-g-culanda@alticelabs.com

Phone

960000004

Account Id

vadson-g-culanda

Figure 4.42: User Account Screen

4.9 User Account Screen Implementation

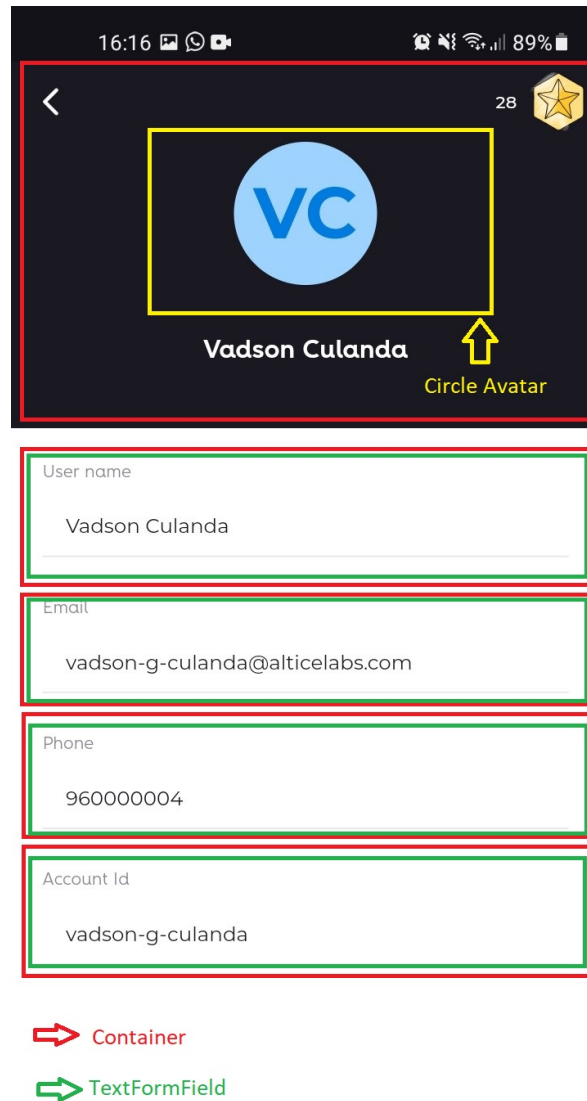


Figure 4.43: UserAccount Screen with labels

In this part there will be a brief description of some of the most relevant widgets used in the screen of figure 4.43. Like in the previous screens we use a **Scaffold**. The “appBar” part contains a back arrow icon that allows to navigate back to the previous screen, a text with the number of smart points and an SVG image of what is a smarty(this is a type of file that is provided by the design team, it’s part of the requirements). On the body of the scaffold we have a “BlocBuilder”(BlocBuilder<UserAccountPageCubit, UserAccountPageStatet>). The BlocBuilder exposes a builder that handles building a widget in response to new states. It is analogous to StreamBuilder, but has simplified API to reduce the amount of boilerplate

code needed as well as cubit-specific performance improvements.

- The “builder” returns a widget based on the `UserAccountPageCubit` state and it’s where the UI is implemented.

```
child: BlocBuilder<UserAccountPageCubit, UserAccountPageState>(
  builder: (context, state) {
    if (state.userAccountModel.phoneNumber != null)
      phoneNumberController.text = state.userAccountModel.phoneNumber;
    if (state.userAccountModel.userName != null)
      userNameController.text = state.userAccountModel.userName;
    if (state.userAccountModel.accountId != null)
      accountIdController.text = state.userAccountModel.accountId;
    if (state.userAccountModel.email != null)
      emailController.text = state.userAccountModel.email;
```

Figure 4.44: `UserAccountPage` excerpt

In the excerpt of code on the fig. 4.44 we initialized the `TextEditingController`s with the information obtained through the `jwt` token (JSON Web Token) (figure 4.45), which is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed.

```
class UserAccountPageCubit extends Cubit<UserAccountPageState> {
  final UserAccountRepository userAccountRepository;
  UserAccountModel userAccountModel;
  UserAccountPageCubit({@required this.userAccountRepository, @required this.userAccountModel})
    : super(UserAccountPageState.initial()) {
    _getUserInfo();
  }

  void _getUserInfo() async {
    userAccountModel = userAccountRepository.getUser();
    emit(state.copyWith(userAccountModel: userAccountModel));
  }
}
```

Figure 4.45: `UserAccountCubit` excerpt

In the cubit page we populate the `UserAccountModel` with the information that was encoded in the `jwt` token and was decoded in the `UserAccountRepository` using the `flutter` package “`jwt_decode`” (figure 4.46 to 4.47).

```
@freezed
abstract class UserAccountModel implements _UserAccountModel {
  const UserAccountModel._();
  const factory UserAccountModel({
    String userName,
    String email,
    String phoneNumber,
    String accountId,
  }) = _UserAccountModel;

  factory UserAccountModel.fromJson(Map<String, dynamic> json) {
    return UserAccountModel(
      userName: json['name'],
      email: json['email'],
      phoneNumber: json['phone'],
      accountId: json['accountId']);
  }
}
```

Figure 4.46: `UserAccountModel` code excerpt

```
class UserAccountRepository {
  UserAccountModel getUser() {
    final payload = Jwt.parseJwt(Session.instance.accessToken);
    return UserAccountModel.fromJson(payload);
  }
}
```

Figure 4.47: UserAccountRepository code excerpt

In the next part of the code we have a Circle Avatar, a widget which is a circle that represents the user's profile image, or, in the absence of such an image, the user's initials. From our account model we have access to the user full name and with a simple function we can extract the user initials and configure the data for building the widget.

In the last part, we simply have a column with four containers that implement a TextFormField, each, and they're populated with the information present in the TextEditingControllers.

4.10 Settings Screen Design

This is a screen where, in the future, the user will be able to change various settings in the App. At the time of the implementation no logic, interactions, or functionalities were defined, therefore this screen is simply a UI screen.

In the figure 4.48 we have the design of this screen which aims to meet the requirements F6 and NF6 from the figure 4.3

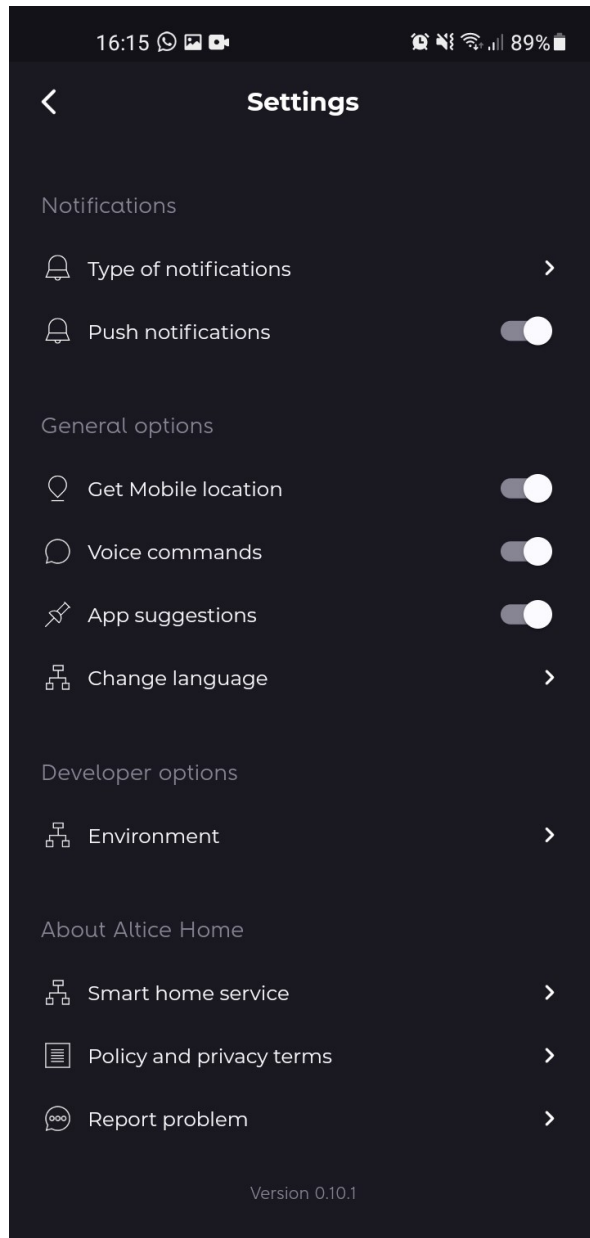


Figure 4.48: Settings Screen

4.11 Settings Screen Implementation

We use a **Scaffold**. The “appBar” part contains back arrow icon that allows us to go back to the previous screen and a text saying “Settings”. On the body of the scaffold we have a “BlocBuilder” (`BlocBuilder<SettingsPageCubit, SettingsPageState>`).

- The “builder” returns a widget based on the `SettingsPageCubit` state and it’s where the UI is implemented.

```
class SettingsItem extends StatefulWidget {
  final String leading;
  final bool switchValue;
  final String text;
  final Function(bool value) onTap;
  final bool isSwitchButton;

  const SettingsItem({
    Key key,
    @required this.leading,
    this.switchValue,
    @required this.text,
    this.onTap,
    @required this.isSwitchButton,
  }) : super(key: key);

  @override
  _SettingsItem createState() => _SettingsItem();
}
```

Figure 4.49: SettingsItem code excerpt

We are using a `Column` widget wrapped around a `SingleChildScrollView`, which is a box in which a single widget that can be scrolled. Inside the `Column`’s children we implemented each element of the list. To avoid boiler plate we created a class called “SettingsItems” where we pass different parameters for each element in the list.

```
void onSwitchNotificationSwitch(bool value) {
  emit(state.copyWith(notificationsSwitch: value));
}

void onSwitchLocationSwitch(bool value) {
  emit(state.copyWith(locationSwitch: value));
}

void onSwitchCommandsSwitch(bool value) {
  emit(state.copyWith(commandsSwitch: value));
}

void onSwitchSuggestionsSwitch(bool value) {
  emit(state.copyWith(suggestionsSwitch: value));
}
}
```

Figure 4.50: SettingsCubit code excerpt

The state of each switch is controlled by the callback function on the settings item, which evokes the cubit whenever the switch is pressed, ending up changing the current state (figure 4.49).

Chapter 5

Testing

5.1 What is testing?

App development is a process that goes through numerous phases. In this part we discuss software testing, which is a crucial part of the development process. A company can not risk to launch a defective product. Therefore, each product must undergo a delicate testing process before being released. The main activities that are included in the testing process are [39, p. 13]:

- design
- planning
- implementation
- execution

5.2 Testing Throughout the Software Development Lifecycle

For each testing process there's an associated software development lifecycle that describes how the activities relate to one another logically and chronologically. Good software testing is characterized by the following characteristics [39, p. 28]:

- For every development activity, there is a corresponding test activity
- Each test level has test objectives specific to that level
- Test analysis and design for a given test level begin during the corresponding development activity
- Testers must always define and review all the requirements and design of the product.

5.2.1 Test levels

Test levels are groups of test activities that are organized and managed together. They can be divided into:

- Component testing

- Integration testing
- System testing
- Acceptance testing

and characterized by the following attributes[39, p. 30]:

- Specific objectives
- Test basis, referenced to derive test cases
- Test object (i.e what is being tested)
- Typical defects and failures
- Specific approaches and responsibilities

5.3 Test management tool

The tool that we used in this project is Xray (see fig. 5.1), which is a full-featured app for Jira that does not require any other software in order to run. Xray supports the entire testing life cycle: test planning, test design, test execution and test reporting.

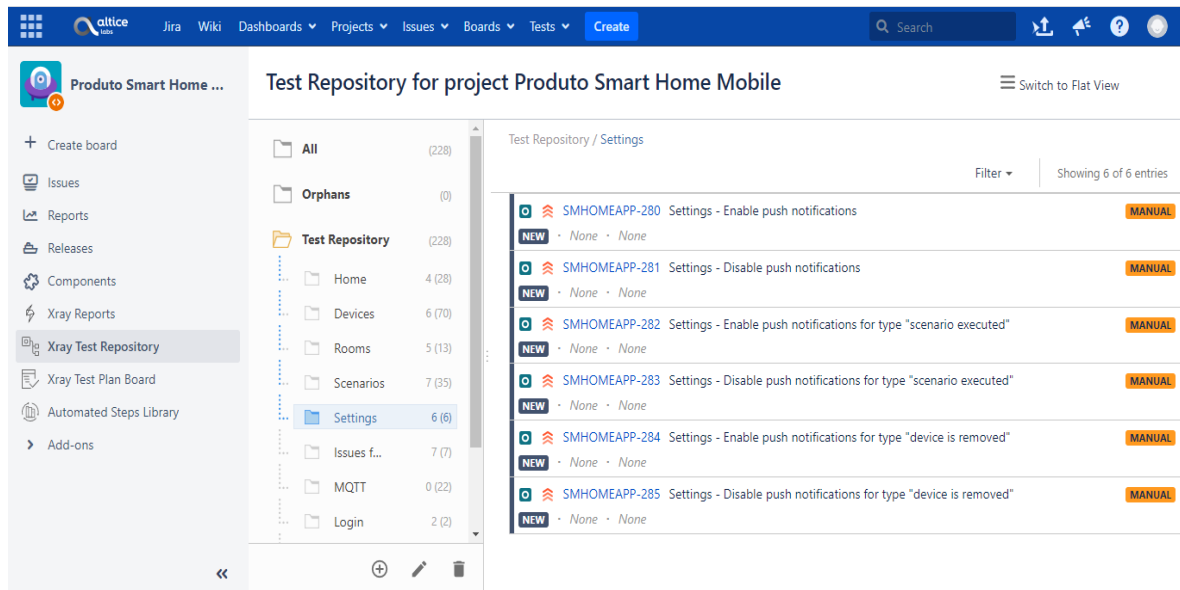


Figure 5.1: Xray layout


5.4 Test Plan

Test planning is a continuous activity and is performed throughout the product's lifecycle [39, p. 66]. Test plans depend on the project and may include different test levels or types. In this project we're doing usability test, namely, acceptance tests and regression tests. To construct a test plan we first need to design a test battery, which will include all the features to validate. Each test from the battery is carefully designed considering the specifications of the App and the Ux planning, and needs to contain all the functionalities we're validating as well as the expected results. The App development is divided into numerous sprints (short period of time where the development team works to complete specific tasks, milestones, or deliverable), each one with a two weeks duration and an associated test plan. In the test plan we always execute three kind of black-box **use case tests***, regression tests, acceptance tests and exploratory tests.

***use case testing** - is a technique that helps identifying test cases that cover the entire system, can be described by preconditions and post-conditions [39, p. 60].

5.4.1 Regression tests

When we start a new sprint where new functionalities of the App are implemented, we need to always execute regression test, to confirm that the changes made, or the defects correction (in case they were defects from the previous sprint) have not caused unforeseen adverse consequences, like accidentally affect the behavior of other parts of the App. This kind of side-effects are called regressions [39, p. 41]. In the image bellow we can see an example of a regression test plan:


Produto Smart Home Mobile / SMHOMEAPP-309

v0.10.0 - Regression- Camera SD Card Playback; Edit Scenarios;

Edit Comment Assign More Reopened

Doc. Generator Export

Details

Type: Test Plan Status: CLOSED (View Workflow)


Priority: Major Resolution: Done/Fixed


Affects Version/s: App-v0.9.0,Cloud-v0.13.0 Fix Version/s: None


Component/s: [Android](#), [App](#), [Cloud](#), [iOS](#)

Labels: None

People

Assignee:  VADSON
GUILHERME AVELINO CULANDA

Reporter:  VADSON
GUILHERME AVELINO CULANDA

Watchers:  Stop watching this issue

Description
Click to add description

Tests

[Test Plan Board](#)
+ Create Test Execution
+ Add

Dates

Created: 2021/05/14 00:41

Updated: 2021/06/02 11:33

Resolved: 2021/05/20 11:49

Figure 5.2: Regression testing

Overall Execution Status **Agile**
[View on Board](#)

9 PASS

Total Tests: 9

[Filter\(s\)](#)

Show entries All Environments Columns

Key	Summary	Requirements	#Test Executions	Issue Assignee	Latest Status
<input type="checkbox"/> SMHOMEAPP-36	Devices - Details - Control - RGB Light - Handle Brightness		2	Vasco Rafael da Graça Coutinho	PASS
<input type="checkbox"/> SMHOMEAPP-37	Devices - Details - Control - RGB Light - Handle Color		2	Vasco Rafael da Graça Coutinho	PASS

Figure 5.3: Regression testing

SMHOMEAPP-10 Onboarding - Customization - Setup multiple devices with addition of room and association to distinct rooms 2 Vasco Rafael da Graça Coutinho PASS ...

Showing 1 to 9 of 9 entries First Previous **1** Next Last

Test Executions Add Test Executions

Show 10 entries Columns

Key	Summary	#Tests	Issue Assignee	Status	
<input type="checkbox"/> SMHOMEAPP-311	v0.9.0 -Android- Regression- Camera SD Card Playback; Edit Scenarios;	9	VADSON GUILHERME AVELINO CULANDA	PASS	...
<input type="checkbox"/> SMHOMEAPP-310	v0.9.0 -iOS- Regression- Camera SD Card Playback; Edit Scenarios;	9	VADSON GUILHERME AVELINO CULANDA	PASS	...

Figure 5.4: Regression testing

5.4.2 Acceptance tests

Acceptance tests are made for the purpose of validating the product and guaranteeing its readiness for deployment and use by the customer. During these tests, defects may be found, but finding them is often not an objective, and finding a significant number of defects during acceptance testing may in some cases be considered a major project risk [39, p. 36]. For every sprint an acceptance test was made, which contained a test battery with the test of the screens or functionalities we wanted to validate.

5.4.3 Exploratory tests

Exploratory tests are informal and undocumented tests that are useful as a complement of the more formal tests. They are a good way to navigate through the app and find some aspects that might need to be improved. They are also beneficial for when there are few or inadequate specifications or significant time available [39, p. 61].

5.5 Validation

The tables in figures 5.5, 5.6 and 5.7 are related to the acceptance tests we executed to validate the functionalities implemented, UI design, and correctness of some strings. These tests were use cases and manually implemented. In the end, all the tests were conducted successfully. During the development and testing at each sprint, if some bug or malfunction was discovered through tests a defect was opened and the problem was fixed. That is the reason why in the final version everything was working correctly.

1 test				
Screens	Name	Pre-condition	Action	Post-condition
Rooms	Access to Rooms Page	Being in the Home menu	Press The rooms options	Be redirected to a screen with a list of all availables rooms
Login	Login Page	Being in the login page	1-Insert credential and press enter 3-Press the eye icon	1- In case the credentials are valid, redirect to the home page 2- In case the credentials are incorrect, show error message 3- Hide or show the password based on the current test
Settings	Settings page	Being in the Home menu	Press the settings option	Be redirected to a screen with a list of various options
User Account	User account	Being in the Home menu	Press the user account icon	Be redirected to a screen with all of the user personal information

Figure 5.5: Table of tests executed in all of the screens

2test			
Name	Pre-condition	Action	Post-condition
Add new room	Being in the rooms page	1-Press "add new room" button 2- Select a room	1-Be redirected to a screen with a list of all availables rooms 2- Only the selected room should be highlighted

Figure 5.6: Table of test executed for the "Add New Room" screen

3 test			
Name	Pre-condition	Action	Post-condition
Choose devices	Being in the Add new room page	Press "add device button	1-Be redirected to a screen with a list of all available devices

Figure 5.7: Table of test executed for the "Choose Devices" screen

As previously indicated, for every screen implemented in the scope of this dissertation there was a thorough process of code review. This process was supported by the Bitbucket version control software, which allows to efficiently work as a team on a shared codebase. We had a main branch called "develop" and for every new functionality or screen developed, we worked on separate branches. After finishing implementing the screen, a pull request was created to the main branch that was merged once approved. An example of the code review can be found in figure 5.8. Because of this code review process a lot of bugs could be prevented before the commit of the new functionality in the App. But most of the issues found were typos related to the implemented screens or mismatches with the design specification, like the size of some icons or images, or the spacing between components.

On the other hand, when testing other functionalities of the App some more relevant issues were found, and we used Jira to document them. This process consists of a brief description of the bug, followed by a screenshot or screen capture of the bug, and finally a register of the version of the equipment used, since some bugs can be related to the software version, in Android or iOS.

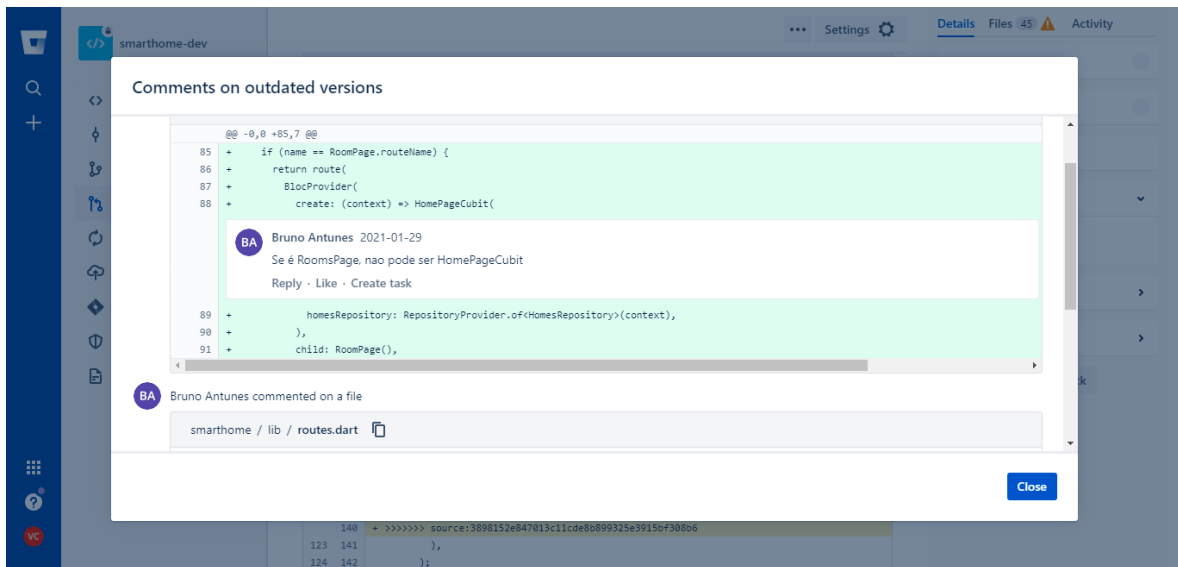


Figure 5.8: Code review of a pull request

Chapter 6

Conclusion

6.1 Conclusion

Smart home technology is a “world” that has been attracting more and more attention. When it comes to setting up a Smart Home ecosystem there are various factors to consider. From all the numerous existing solutions, like Google Assistant, Alexa, Apple Home app, Samsung SmartThings, among others, is difficult to define the best ecosystem. Most of these solutions offer similar benefits and each one excels in some particular area(s). Therefore, the best solution depends on the user preferences. This dissertation had the objective of developing a mobile App in Flutter for smart home management that could rival those solutions and offer something more.

The used approach was focused on targeting the customer desire of having a user-friendly experience at home, using technologies that simplify the set up and management of smart devices in the house. Flutter has only been around for 4 years, but throughout this project, we noticed that we’re able to achieve a lot with it. Even if some setbacks were encountered, there a lot of libraries available that are useful and play a major role in the development.

When implementing all the screens we learned about the importance of defining the use-cases and the requirement because they allow a better understanding of what are the client needs. We also learned about some of the best practices we should have in mind to increase performance, track the changes more efficiently, have stricter control over the data (by using immutable variables), and avoid unnecessary portions of repeated code (by using global classes). Lastly, we proceeded to the validation part where we executed a number of acceptance tests, guaranteeing that all the requirements were met as intended.

In a final note with this project, we were able to learn all the steps and phases behind the development of a mobile App and the testing process that occurs before releasing the product. There are a lot of test techniques that could be used, but each of them has a different efficiency depending on the product. For this project, we learned what was the best to use and how to design and execute the tests.

6.2 Future Works

The Altice Home app is in the MVP phase and is being actively developed, which means it still has room for improvement and restructuring. Presently, the app only supports Tuya devices such as RGB/white lights, plugs and cameras, but the goal for the future is to support even more devices like air purifiers, locks, or thermostats and also devices from other brands and manufacturers. There's also a number of functionalities in the App that aren't currently implemented. The future goal is for the app to evolve to other areas of interest, such as Entertainment, Connectivity, Health, Security and related domains. The end vision is to have a Smart Home ecosystem that allows for a unified experience in a friendly and unique interface to control all the devices and systems at home.

Some of the envisioned advanced functionalities are the following:

- **Multiple homes and users** - The app will support multiple homes for a single user (for example the house and the work office) and the possibility of inviting people (typically family members) to our house and with the options of defining the role;
- **ChatBot** - The Bot will be like a virtual assistant it will be particular skilled for in-home automation services and the idea is to support the most common interaction for controlling the house, either texting or using voice directly in the Chat Bot widget;
- **Alerts and push notifications** - Instant notifications to the smart device to keep the user aware of whats happening at home;
- **Device and scenario history** - The app will provide a monthly energy consumption graph and daily updates for when the devices and scenarios status gets updated
- **Planned Activity** - The app will allow the user to plan his day, week or month, by scheduling a series of events and scenarios;

Bibliography

- [1] Vincent Ricquebourg, David Menga, David Durand, Bruno Marhic, Laurent Delahoche, and Christophe Logé. The smart home concept: Our immediate future. *2006 1st IEEE Int. Conf. E-Learning Ind. Electron. ICELIE*, pages 23–28, 2006. doi:10.1109/ICELIE.2006.347206.
- [2] Othmar Kyas. *How To Smart Home*, volume 3. 2013.
- [3] Smart Nutter. Mesh Wi-Fi Router for Smart Home Automation – SmartNutter. URL: <https://smarnutter.com/mesh-wi-fi-router-for-smart-home-automation/>.
- [4] Brian Heater. ALEXA gets access to Wolfram’s Alpha knowledge engine, 2018. URL: <https://techcrunch.com/2018/12/20/alexa-gets-access-to-wolfram-alphas-knowledge-engine/>.
- [5] John Walsby and Megan Stewart. Guidelines for Amazon Echo Alexa brand usage for external partners. pages 1–48, 2019.
- [6] Amazon. Works with Alexa IoT Solution Providers — Alexa Skills Kit, 2010. URL: <https://developer.amazon.com/en-US/docs/alexa/smarthome/smart-home-iot-solution-providers.html>.
- [7] Home Assistant. Home Assistant vs. Home Assistant Core. URL: <https://www.home-assistant.io/faq/ha-vs-hassio/>.
- [8] Home Assistant. Z-Wave. URL: <https://www.home-assistant.io/docs/z-wave/>.
- [9] Home Assistant. MQTT. URL: <https://www.home-assistant.io/integrations/mqtt/>.
- [10] Home Assistant. Automation Trigger. URL: <https://www.home-assistant.io/docs/automation/trigger>.
- [11] Home Assistant. Automation Conditions. URL: <https://www.home-assistant.io/docs/automation/condition>.
- [12] Home Assistant. Automation Actions. URL: <https://www.home-assistant.io/docs/automation/action>.
- [13] Home Assistant. Scenes. URL: <https://www.home-assistant.io/integrations/scene/>.

- [14] Samsung. SmartThings Developers, 2020. URL: <https://smarthings.developer.samsung.com/docs/platform-basics.html><https://smarthings.developer.samsung.com/docs/index.html>.
- [15] Samsung. SmartThings Developers — Documentation, 2020. URL: <https://smarthings.developer.samsung.com/docs/devices/device-basics.html><https://smarthings.developer.samsung.com/docs/index.html>.
- [16] Samsung. Use automations in SmartThings. URL: <https://www.samsung.com/us/support/answer/ANS00078852/>.
- [17] Rixin Xu, Qiang Zeng, Liehuang Zhu, Haotian Chi, Xiaojiang Du, and Mohsen Guizani. Privacy Leakage in Smart Homes and Its Mitigation: IFTTT as a Case Study. *IEEE Access*, 7:63457–63471, 2019. arXiv:1902.03168, doi:10.1109/ACCESS.2019.2911202.
- [18] Samsung. SmartThings Developers — Documentation, 2020. URL: <https://smarthings.developer.samsung.com/docs/devices/working-with-scenes.html><https://smarthings.developer.samsung.com/docs/index.html>.
- [19] Nicholas Fifield. Apple HomeKit Application and Cost Breakdown. 2020.
- [20] Apple. HomeKit Overview - Apple Developer, 2021. URL: <https://developer.apple.com/homekit/>.
- [21] James Stables. Apple HomeKit: Everything you need to know about living in an Apple Home, 2021. URL: <https://www.the-ambient.com/guides/apple-homekit-complete-guide-194>.
- [22] Apple. HomeKit Accessory FAQs - HomeKit - Apple Developer, 2021. URL: <https://developer.apple.com/homekit/faq/>.
- [23] Apple. Add a HomeKit accessory to the Home app - Apple Support, 2021. URL: <https://support.apple.com/en-us/HT204893#room>.
- [24] Apple. Terminology and Layout - HomeKit - Human Interface Guidelines - Apple Developer, 2021. URL: <https://developer.apple.com/design/human-interface-guidelines/homekit/overview/terminology-and-layout/>.
- [25] Apple. iOS - Home - Apple, 2021. URL: <https://www.apple.com/ios/home/>.
- [26] Gadget Freak. Starting With Tuya: The Ultimate Guide! - Gadget-Freakz.com, 2020. URL: <https://gadget-freakz.com/starting-with-tuya-the-ultimate-guide/>.
- [27] Tuya. What is the Difference between Native App and Hybrid App? — Existek Blog, 2021. URL: https://support.tuya.com/en/help/_detail/K9hrdz10p224e<https://existek.com/blog/difference-between-native-app-and-hybrid-app/>.
- [28] Tuya. What does Tuya Smart App do? What are the advantages?, 2021. URL: https://support.tuya.com/en/help/_detail/Kajju51fy56w3.
- [29] Tuya Inc. What’s the advantages of Tuya, 2021. URL: https://support.tuya.com/en/help/_detail/K8sdy0ybarg3y.

- [30] Tuya. OEM App-Docummentation-Tuya Developer, 2021. URL: <https://developer.tuya.com/en/docs/iot/oem-app?id=K9j6yqpc43u9t>.
- [31] Tuya. Glossary-Docummentation-Tuya Developer, 2021. URL: <https://developer.tuya.com/en/docs/iot/terms?id=K914jqo6tegj4#title-2-Communication>.
- [32] Sinclair,Patrick. HomeKit vs SmartThings Protocols: Which to Choose? — All Home Robotics. URL: <https://www.allhomerobotics.com/homekit-vs-smartthings-protocols-which-to-choose/>.
- [33] Christian De Looper. Google Assistant vs Amazon Alexa vs HomeKit: which smart home is best? - Business Insider, 2020. URL: <https://www.businessinsider.com/homekit-vs-google-assistant-vs-amazon-alexahttps://www.businessinsider.com/homekit-vs-google-assistant-vs-amazon-alexa?r=US&IR=T>.
- [34] Amazon. Alexa for Device Makers. URL: <https://developer.amazon.com/en-US/alexa/devices>.
- [35] Google. Data security and privacy on devices that work with Assistant - Google Nest Help. URL: <https://support.google.com/googlenest/answer/7639952?hl=enhttps://support.google.com/googlenest/answer/7072285?hl=en>.
- [36] Visual Paradigm. What is Use Case Diagram? URL: <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/>.
- [37] Qra corp. Functional vs Non-Functional Requirements: The Definitive Guide. URL: <https://qracorp.com/functional-vs-non-functional-requirements/>.
- [38] geeksforgeeks. Why is Immutability so Important in JavaScript? URL: <https://www.geeksforgeeks.org/why-is-immutability-so-important-in-javascript/>.
- [39] International Software Testing Qualifications Board. Certified Tester Foundation Level Syllabus. page 85, 2011.

