TECNOLOGIA SETÚBAL
ESCOLA SUPERIOR
POLITÉCNICO SETÚBAL

ADMILTON KENDY CONCEIÇÃO GOMES

SISTEMA DE LOCALIZAÇÃO POR UWB

Relatório de Dissertação como requisito parcial para obtenção do grau de Mestre em Engenharia Eletrotécnica e de Computadores.

ORIENTADOR

Doutor, Rui Antunes

Dezembro 2024

ADMILTON KENDY CONCEIÇÃO GOMES

SISTEMA DE LOCALIZAÇÃO POR UWB

JÚRI

Presidente: Doutor, Filipe Cardoso, ESTSetúbal/IPS
Orientador: Doutor, Rui Antunes, ESTSetúbal/IPS
Arguente: Doutor Luís Palma, DEEC/FCT-NOVA

Dezembro 2024

Agradecimentos

Gostaria de agradecer, em primeiro lugar ao meu orientador, Professor Doutor Rui Antunes, por toda a sua orientação, paciência e apoio durante o desenvolvimento desta dissertação, bem como pelo acompanhamento, fornecimento de material, motivação, partilha de conhecimentos, amizade e incentivo dado à minha autonomia.

Agradeço a todos que me apoiaram durante a realização deste trabalho. Um agradecimento especial à minha filha, Elisa Gomes, cuja alegria e presença constante foram uma fonte de motivação. Agradeço também à minha irmã Kelly Gomes, pelo apoio incondicional e pelas palavras de encorajamento ao longo desta jornada. Por fim, agradeço a todos os que, de alguma forma, contribuíram para a concretização deste projeto.

Resumo

Esta dissertação apresenta o desenvolvimento de um sistema inovador de medição de distâncias baseado em tecnologia *Ultra-Wideband* (UWB) e comunicação *Bluetooth Low Energy* (BLE), destinado a fornecer medições de alta precisão em tempo-real através de uma aplicação móvel Android. A solução consiste em dois módulos principais: uma *Tag* móvel e uma Âncora fixa, ambos equipados com o módulo DW1000 e a placa XIAO ESP32C3, que facilitam a medição precisa de distâncias e a transmissão dos dados para um dispositivo móvel. A metodologia adotada inclui o desenvolvimento de *hardware*, a implementação do *firmware* para gestão das comunicações UWB e BLE, e o projeto da aplicação móvel, criando um sistema de fácil utilização.

Os testes realizados demonstraram estabilidade na comunicação BLE e medições precisas, com resolução aos centímetros, até uma distância de 12 metros, confirmando a eficácia do sistema desenvolvido. Este trabalho contribui para o avanço no campo dos sistemas de localização e monitorização, com potencial para aplicações em diversas áreas associadas à Indústria 4.0 e 5.0. A solução proposta reflete os princípios da Indústria 4.0, ao integrar tecnologias digitais e conexão de dispositivos para aplicações como a logística, navegação interna e rastreamento em tempo real. Além disso pode ser utilizado na indústria 5.0 nas interfaces humano-máquina.

Palavras-chave: Medição de distância, *Ultra-Wideband*, *Bluetooth Low Energy*, *Tag* e Âncora, dispositivos móveis, microcontrolador RISC.

Abstract

This dissertation presents the development of an innovative distance measurement system based on Ultra-Wideband (UWB) technology and Bluetooth Low Energy (BLE) communication, designed to provide high-precision measurements in real-time through an Android mobile application. The solution consists of two main modules: a mobile Tag and a fixed Anchor, both equipped with the DW1000 module and the XIAO ESP32C3 board, which facilitate accurate distance measurement and data transmission to a mobile device. The methodology adopted includes hardware development, firmware implementation for managing UWB and BLE communications, and mobile application design, creating an easy-to-use system.

The tests carried out demonstrated stability in BLE communication and precise measurements, with centimeter resolution, up to 12 meters, confirming the effectiveness of the developed system. This work contributes to advancements in the field of localization and monitoring systems, with potential applications in various areas associated with Industry 4.0 and 5.0. The proposed solution reflects the principles of Industry 4.0 by integrating digital technologies and device connectivity for applications such as logistics, indoor navigation, and real-time tracking. Furthermore, it can be employed in Industry 5.0 for human-machine interfaces.

Keywords: Distance measurement, Ultra-Wideband, Bluetooth Low Energy, Tag and Anchor, mobile devices, RISC microcontroller.

Índice

Lista de Figuras

Lista de Tabelas

Lista de Siglas e Acrónimos

BLE	*Bluetooth Low Energy*
FCC	*Federal Communications Commission*
GPS	*Global Positioning System*
IoT	Internet das Coisas
RF	*Radio Frequency*
RFID	*Radio Frequency Identification*
RTLS	*Real-Time Locating System*
SNR	*Signal-to-Noise Ratio*
SPI	*Serial Peripheral Interface*
ToF	*Time of Flight*
UWB	*Ultra-Wideband*
WiFi	*Wireless Fidelity*

Capítulo 1
Introdução

1.1. Enquadramento e motivação

Este trabalho foi desenvolvido no âmbito do Mestrado em Engenharia Eletrotécnica e de Computadores, com especialização em Computadores e Sistemas Ciberfísicos. Com o avanço da Internet das Coisas (IoT) e o crescimento de aplicações que requerem localização precisa, a necessidade de sistemas que combinem elevada precisão com baixo consumo energético torna-se evidente (Zafari et al., 2019; Inpixon, 2024). A tecnologia UWB tem-se destacado neste contexto devido à sua capacidade de fornecer medições de alta precisão em ambientes interiores, atravessando obstáculos com eficiência e operando com baixa potência de transmissão. Por outro lado, a tecnologia BLE complementa essa abordagem ao permitir comunicações eficientes e de baixo consumo energético com dispositivos móveis, sendo amplamente utilizada em aplicações IoT (Mullner et al., 2020; Decarli et al., 2021; Fischer e Marzullo 2019; Gomez et al., 2012).

A integração entre UWB e BLE apresenta-se como uma solução inovadora para sistemas de localização em tempo-real, onde cada tecnologia desempenha um papel essencial. A tecnologia UWB é responsável por medir distâncias com elevada precisão através de impulsos curtos e controlados pelo tempo de voo (ToF), enquanto a BLE oferece a plataforma de comunicação necessária para a transferência de dados para dispositivos móveis (Yang et al., 2020).

Na figura 1 é apresentada uma breve comparação entre a tecnologia UWB e outras tecnologias.

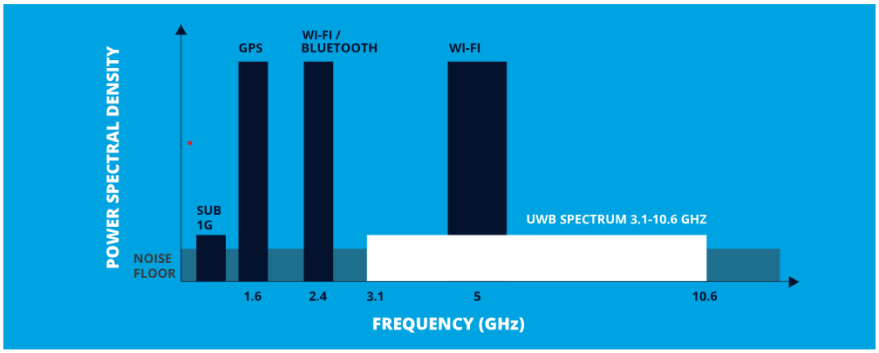

Figura 1- Comparação da densidade espectral de potência entre várias tecnologias de comunicação (Inpixon, 2023).

Os sistemas de localização em tempo-real (RTLS) têm encontrado aplicações numa ampla variedade de campos, incluindo logística, segurança, saúde, rastreamento de ativos e entretenimento. No entanto, a maioria das soluções existentes enfrenta limitações relacionadas com a precisão, eficiência energética e custo (Inpixon, 2023; IEEE Communications Society, 2018). O sistema proposto

neste trabalho procura superar estas limitações ao combinar a precisão da tecnologia UWB com a flexibilidade e o baixo consumo energético da comunicação BLE (Gallagher & Detweiler, 2013). A aplicação desenvolvida na ferramenta Android Studio funcionará como interface do utilizador, apresentando os resultados num ambiente gráfico acessível e funcional. A figura 2 apresenta a arquitetura do sistema desenvolvido.

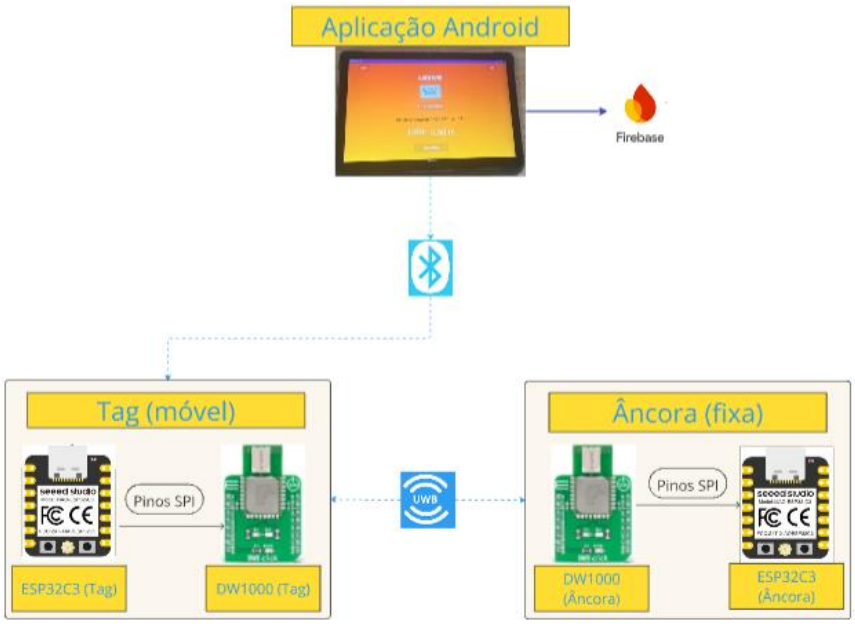

Figura 2 – Arquitetura do sistema desenvolvido.

1.2. Objetivos

O trabalho tem como principal objetivo desenvolver um sistema eletrónico que integre as tecnologias UWB e BLE para localização em tempo-real, proporcionando medições precisas em diversos cenários experimentais. Pretende-se também criar uma aplicação móvel intuitiva para exibir e interagir com os dados recolhidos, bem como validar o desempenho do sistema em comparação com outras tecnologias, como GPS, Wi-Fi e RFID, evidenciando as suas vantagens no contexto atual de localização e rastreamento.

1.3. Contribuições originais

O presente trabalho apresenta contribuições originais no campo dos sistemas de localização em tempo-real, destacando-se:

A seção 3.3, Arquitetura do *hardware*, apresenta a arquitetura do sistema de *hardware* desenvolvido, incluindo a estrutura dos componentes principais, a comunicação entre eles e a integração com o sistema de *software*; As seções 4.2 e 4.3 detalham o desenho do esquema no Schematic Editor, com o processo de conceção das Placas de Circuito Impresso (PCBs) para o sistema de medição, utilizando a ferramenta freeware KiCad, abrangendo as etapas de conceção do esquema elétrico, o *layout* da PCB e considerações específicas para a integração dos módulos XIAO ESP32C3; O capítulo 5 apresenta o desenvolvimento da aplicação, incluindo a aplicação móvel concebida na ferramenta Android Studio, que comunica com a *Tag* utilizando a tecnologia BLE integrada no módulo XIAO ESP32C3, detalhando o processo desde a criação da interface até à implementação das funcionalidades essenciais de comunicação; O capítulo 6 descreve os testes realizados ao sistema de medição de distâncias baseado na tecnologia UWB.

1.4. Estrutura da dissertação

Esta dissertação encontra-se organizada em sete capítulos principais. O Capítulo 1 apresenta uma introdução geral ao tema, descrevendo os objetivos principais, o enquadramento e as motivações, as contribuições originais e a estrutura do trabalho. O Capítulo 2 aborda um enquadramento ao estudo da tecnologia UWB, os seus princípios de funcionamento e a comparação com outras tecnologias de localização. O Capítulo 3 foca-se no desenvolvimento do *hardware*, abordando a escolha dos componentes, a arquitetura do sistema e os aspetos de integração entre os módulos UWB e BLE.

No Capítulo 4, é descrito o desenho e desenvolvimento das placas de circuito impresso (PCBs), utilizando ferramentas como o KiCad, abrangendo desde a criação do esquema elétrico até à montagem final. O Capítulo 5 trata do desenvolvimento da aplicação Android, explicando os detalhes da sua arquitetura, interfaces e funcionalidades implementadas.

Os testes ao sistema e a avaliação dos resultados são discutidos no Capítulo 6, incluindo análises de desempenho, precisão e validação experimental. Finalmente, o Capítulo 7 apresenta as conclusões do trabalho, destacando as contribuições obtidas e propondo direções para futuras investigações na área.

Capítulo 2

Estudo da tecnologia UWB

2.1. Introdução à tecnologia UWB

A tecnologia UWB destaca-se no campo das comunicações *wireless* e dos sistemas de localização devido à sua elevada precisão e capacidade de transmitir dados em frequências amplas (Molisch et al., 2006), com menos interferência de outros dispositivos, melhorando assim a segurança e eficiência (Zafari et al., 2019; Zwirello e Schipper 2013). A tecnologia UWB transmite sinais de rádio (figura 1), permitindo fazer medição de distância com precisão e com baixa potência de transmissão, sendo ideal para aplicações de baixo consumo energético.

Entre as principais vantagens da tecnologia UWB, destaca-se a capacidade de transmitir pequenas quantidades de dados a curta distância, com um consumo mínimo de energia, o que a torna apropriada para ser incorporada em dispositivos eletrónicos de baixa potência, nomeadamente em sensores para ambientes de IoT (Internet das Coisas). Entre as aplicações mais relevantes da tecnologia UWB, destacam-se os sistemas de localização em tempo-real (RTLS), de comunicações de curta distância e as redes de sensores sem fio (Alarifi et al., 2016; Decawave, 2020). Além disso, esta tecnologia tem mostrado o seu vasto potencial em aplicações industriais e comerciais (Geng et al., 2014), como por exemplo no rastreamento, em sistemas biomédicos e no controlo de acessos. Recentemente, a empresa Apple criou o sistema AirTag, permitindo o rastreamento e a localização de objetos em tempo-real (Decarli et al., 2021 e Apple, 2021).

2.2. Princípio de funcionamento

A tecnologia UWB é eficaz na localização em tempo-real, funcionando com a transmissão de impulsos curtos e de baixa potência, capazes de penetrar em ambientes e obstáculos, de forma a proporcionar medições precisas de distância entre pontos (Mullner et al., 2020). O espetro utilizado pela tecnologia UWB segue o regulamento da *Federal Communications Commission* (FCC), estando compreendido entre 3,1 GHz e 10,6 GHz, pois nesta faixa de transmissão é garantida a não interferência com outros sistemas de comunicação e a transferência de dados para aplicações de localização e comunicação (Yang et al., 2020).

O funcionamento da tecnologia UWB na prática é diferente das tecnologias tradicionais, já que enquanto nas tradicionais são utilizadas ondas contínuas, a tecnologia UWB envia impulsos rápidos, medindo o tempo de voo (ToF) dos impulsos e determinando a distância entre dispositivos (Zwirello et al., 2012; Fontana & Gunderson, 2002). Na figura 3 é apresentado o diagrama de sequência de comunicação UWB entre dois dispositivos.

A baixa potência de transmissão da tecnologia UWB permite atravessar obstáculos, sendo adequada para utilização em ambientes internos e em sistemas que exigem alta segurança, como por

exemplo, nas aplicações financeiras ou nos sistemas de defesa.

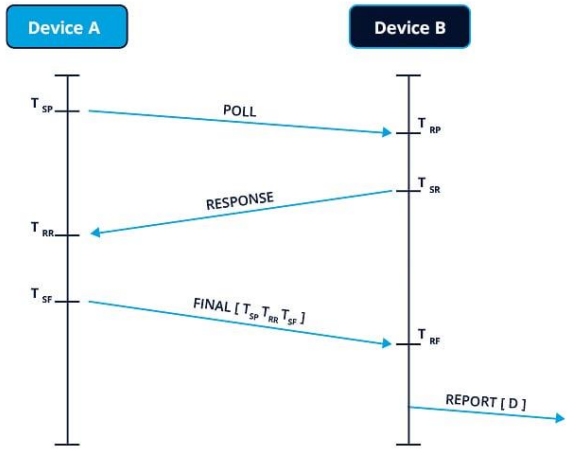

Figura 3 – Diagrama de sequência de comunicação entre dois dispositivos UWB (Inpixon, 2023).

A Figura 3, apresenta uma comunicação bidirecional entre dois dispositivos para medir a distância entre eles. Quando um dispositivo está próximo do outro, ambos iniciam uma troca de informações para determinar essa distância, mesmo enquanto comunicam dados entre si. O tempo que o sinal demora a percorrer o espaço entre os dispositivos é multiplicado pela velocidade da luz, permitindo assim calcular a sua posição relativa. Esta informação de posição pode, em muitos casos, ser utilizada para comunicação com reconhecimento de localização.

A localização obtida pode ser aproveitada conforme a aplicação específica, sendo possível recorrer a âncoras fixas e dispositivos UWB. No entanto, o sistema só consegue utilizar um dispositivo de referência (alcance) de cada vez para localizar um dispositivo específico.

2.3. Alcance e exatidão da tecnologia UWB

A tecnologia UWB pode localizar um dispositivo com levada precisão em distâncias de até 200 metros. No entanto, trabalha com mais eficiência em curtas distâncias, geralmente entre 1 e 50 metros, e funciona melhor com linha de visão entre dispositivos ou âncoras. Dentro deste intervalo, a tecnologia UWB fornece comunicações altamente precisas, com exatidão entre 10 e 30 centímetros, rápidas e seguras, e é altamente imune a interferências (Inpixon, 2023).

2.4. Comparação com outras tecnologias de localização

A tecnologia UWB comparada com outras tecnologias de localização como o GPS, Wi-Fi, BLE e RFID possuiu algumas vantagens. Em termos de comparação, a tecnologia UWB possui mais precisão em ambientes fechados do que, por exemplo, o GPS, sendo este último limitado devido à necessidade de comunicação direta com satélites (Decarli et al., 2021; Mullner & Wieser, 2019). Devido à medição do ToF dos impulsos, a tecnologia UWB possui maior precisão, na gama dos centímetros, do que o Wi-Fi e BLE, porque estes utilizam metodologias de estimação mais sensíveis a interferências, baseadas na intensidade de sinal, para medir distância (Zafari et al., 2019; Bartholomew & Green, 2019; Lemic et al., 2016).

No que diz respeito ao RFID, devido à faixa de operação ampla e a não ser necessário os dispositivos estarem na linha de visão direta, faz com que a tecnologia UWB tenha melhor desempenho em ambientes complexos, conseguindo-se um melhor alcance e precisão. Devido à flexibilidade da tecnologia UWB, torna-se assim possível a sua aplicação em diferentes cenários, como a medição de produtos industriais em depósito (Mullner et al., 2020; Gezici et al., 2005).

Por fim, a tecnologia UWB destaca-se entre as demais tecnologias, sendo mais atrativa devido à sua segurança. É difícil interferir na comunicação UWB, consistindo por isso numa boa alternativa para aplicações que exigem um elevado nível de segurança, nomeadamente no controlo de acessos e nos dispositivos de pagamento.

Tabela 1 - Tabela comparativa com outras tecnologias (Inpixon, 2023).

Característica	UWB	BLE	Wi-Fi	Zigbee	LoRa	NFC	RFID
Custo	Elevado	Reduzido	Moderado/ Elevado	Baixo	Muito baixo	Baixo	Variável
Latência	< 1 ms	~30 ms	~10 ms	~50 ms	Alta (segundos)	< 0,1 s	< 0,1 s
Alcance de Transmissão	0-50 m (até 200 m)	Até 75 m	Até 150 m	10-20 m	Até 15 km	Até 0,1 m	Centímetros a metros
Consumo de Energia	Baixo	Baixo	Elevado	Muito baixo	Muito baixo	Muito baixo	Muito baixo
Frequência	3,1-10,6 GHz	2,4 GHz	2,4/5/6 GHz	2,4 GHz	868/915 MHz	13,56 MHz	Várias (kHz a GHz)
Taxa de Dados	Até 27 Mbps	Até 2 Mbps	Até 9,6 Gbps	20-250 kbps	0,3-50 kbps	Até 424 kbps	Variável
Exatidão	10-30 cm	< 5 m	< 10 m	~3 m	~10 m	Alta	Alta

Capítulo 3
Desenvolvimento do *hardware*

3.1. Introdução

Este capítulo representa uma das partes fundamentais do projeto, abordando o desenvolvimento do *hardware* utilizando a tecnologia UWB e a comunicação Bluetooth *Low Energy* para medição da distância entre dispositivos. É apresentada a arquitetura do sistema, os desafios encontrados e as soluções adotadas, as razões da escolha dos componentes utilizados para o desenvolvimento do *hardware* e a discussão dos resultados experimentais.

O sistema desenvolvido é constituído por dois módulos: o dispositivo móvel, a *Tag*; e o fixo, a âncora. Esses módulos eletrónicos são construídos por uma placa XIAO ESP32C3 (custa 7.40 € na Mouser), em conjunto com o módulo UWB DW1000 (custa aproximadamente 91 €), de forma a garantir medições fiáveis.

Este trabalho tem como objetivo criar um sistema eletrónico de localização, capaz de medir distâncias e/ou calcular a localização em tempo-real, utilizando uma Âncora e uma *Tag*. Para atingir esse objetivo foi escolhida a placa microcontroladora XIAO ESP32C3, devido à sua facilidade de comunicação BLE com a App Android. A tecnologia de comunicação BLE permite que os dados de medição da distância sejam transmitidos à aplicação desenvolvida e que o utilizador possa assim visualizá-los de forma fácil e em tempo-real.

O desenvolvimento do *hardware* deu-se em várias etapas: desde a seleção e a integração dos componentes, escolhidos criteriosamente para assegurar a compatibilidade e o desempenho do sistema, de acordo com as especificidades impostas para o desenvolvimento deste projeto, até à implementação do *firmware* e à calibração dos módulos.

Para além do acima supracitado, a discussão dos resultados dos testes efetuados permitem avaliar o desempenho do sistema, com foco na eficácia da comunicação e na precisão das medições. De modo, a exemplificar a integração e transmissão de dados em tempo-real, faz-se uma pequena alusão ao capítulo cinco.

3.2. Componentes utilizados

Nesta secção é descrito o processo de seleção dos componentes utilizados para desenvolver o *hardware*, justificando-se as escolhas técnicas e explicando como os componentes escolhidos contribuem para o funcionamento do sistema de medição.

3.2.1. Placa XIAO ESP32C3

A placa microcontroladora XIAO ESP32C3 foi a escolhida para o desenvolvimento do *hardware*, devido às suas características ideais de integração de comunicação BLE e Wi-Fi, e ainda ao seu baixo custo, permitindo a medição de distâncias e a transmissão dos dados para uma aplicação móvel. Na tabela 2 são descritas as principais características da placa Seeed XIAO ESP32C3, bem como das restantes placas XIAO.

Tabela 2 – Especificações técnicas da placa Seeed XIAO ESP32C3, extraído de Seeed Studio (2023).

Item	Seeed Studio XIAO ESP32C3	Seeeduino XIAO	Seeed XIAO RP2040	Seeed XIAO nRF52840	Seeed XIAO nRF52840 Sense
Processor	ESP32-C3 32-bit RISC-V @160MHz	SAMD21 M0+@48MHz	RP2040 Dual-core M0+@133Mhz	nRF52840 M4F@64MHz	nRF52840 M4F@64MHz
Wireless Connectivity	WiFi and Bluetooth 5 (LE)	N/A	N/A	Bluetooth 5.0/BLE/NFC	Bluetooth 5.0/BLE/NFC
Memory	400KB SRAM, 4MB onboard Flash	32KB SRAM 256KB FLASH	264KB SRAM 2MB onboard Flash	256KB RAM, 1MB Flash 2MB onboard Flash	256KB RAM,1MB Flash 2MB onboard Flash
Built-in Sensors	N/A	N/A	N/A	N/A	6 DOF IMU (LSM6DS3TR-C), PDM Microphone
Interfaces	I2C/UART/SPI	I2C/UART/SPI	I2C/UART/SPI	I2C/UART/SPI	I2C/UART/SPI
PWM/Analog Pins	11/4	11/4	11/4	11/4	11/4
Onboard Buttons	Reset/ Boot Button	N/A	Reset/ Boot Button	Reset Button	Reset Button
Onboard LEDs	Charge LED	N/A	Full-color RGB/ 3-in-one LED	3-in-one LED/ Charge LED	3-in-one LED/ Charge LED
Battery Charge Chip	Built-in	N/A	N/A	BQ25101	BQ25101
Programming Languages	Arduino/ MicroPython	Arduino/ CircuitPython	Arduino/ MicroPython/ CircuitPython		

3.2.1.1. Funcionalidades relevantes para o Projeto

A placa XIAO ESP32C3 é ideal para o desenvolvimento deste sistema de medição/localização 1D, devido às suas funcionalidades de conectividade BLE, que favorece a integração da aplicação móvel e possibilitando futuramente a escalabilidade do sistema. O baixo consumo de energia permite maior autonomia em dispositivos móveis (Gollakota et al., 2013). O desempenho melhorado deve-se à antena externa integrada, que assegura uma comunicação mais estável e de longo alcance, que é essencial para garantir a fiabilidade das medições, a transmissão dos dados e a compatibilidade com Shields Grove (Seeed Studio, 2023). Estas funcionalidades permitem que a placa XIAO ESP32C3 seja uma escolha sólida e versátil para o desenvolvimento de sistemas de medição de distâncias com comunicação sem fios em tempo-real. Na figura 4 é ilustrada a placa Seeed XIAO ESP32C3 e respetivos pinos.

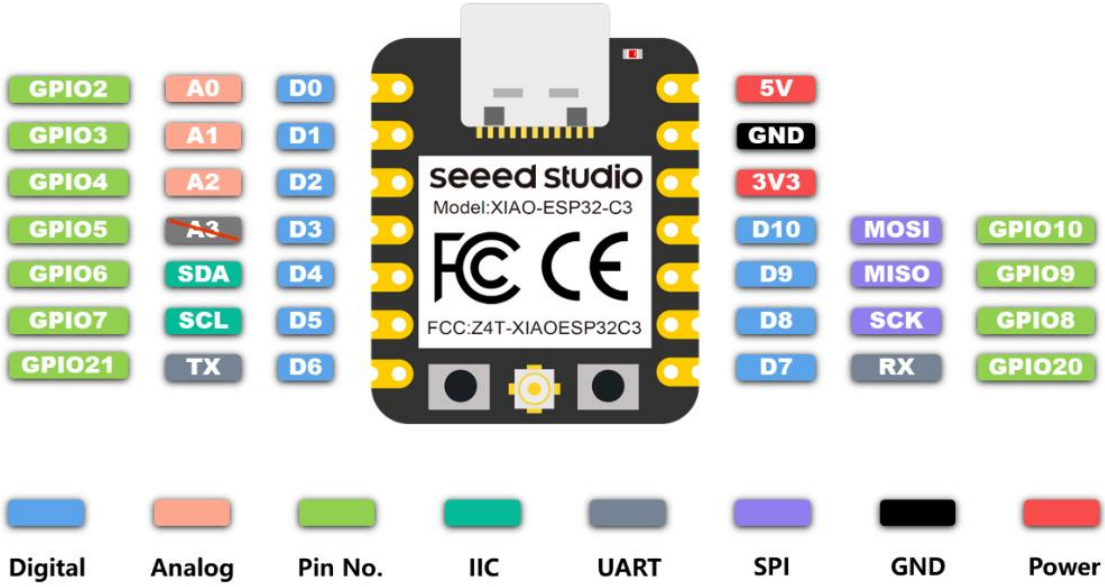

Figura 4 - Diagrama dos pinos da placa XIAO ESP32C3, extraído de Seeed Studio (2023).

O microcontrolador XIAO ESP32C3 é constituído por pinos com diferentes funções, permitindo uma ampla gama de aplicações. Estes pinos estão organizados e identificados por cores, o que facilita a compreensão e utilização.

- Pinos digitais (azul): podem ser configurados como entradas ou saídas digitais, adequados para a aquisição e controlo de sinais binários (D0, D1, D2, D3, D4, D5, D6, D7, D8, D9 e D10).
- Pinos analógicos (laranja): permitem a aquisição de sinais analógicos, comuns em sensores que geram variações contínuas de tensão (A0, A1, A2 e A3).
- Pinos GPIO (verde): são de uso geral e podem ser configurados como entradas ou saídas, dependendo da necessidade da aplicação (GPIO2, GPIO3, GPIO4, GPIO5, GPIO6, GPIO7, GPIO8, GPIO9, GPIO10, GPIO20 e GPIO21).
- Pinos UART (cinzento): usados para comunicação série assíncrona (TX para transmissão

de dados e RX para receção de dados).

- Pinos SPI (roxo), permitem a comunicação de alta velocidade com dispositivos periféricos compatíveis com este protocolo:
  - MOSI (Master Out Slave In): envio de dados do *Master* para o *Slave*.
  - MISO (Master In Slave Out): envio de dados do *Slave* para o *Master*.
  - SCK (Serial Clock): relógio série para sincronização.
- Pinos de alimentação:
  - 5V (vermelho): Pode ser usado para fornecer 5V ao dispositivo ou receber 5V para alimentar o microcontrolador.
  - 3V3 (vermelho-claro): Fornece 3,3V para circuitos de baixa potência.
  - GND (preto): Pinos de terra (*ground*), essenciais para completar o circuito elétrico.

3.2.2. Módulo DW1000

O módulo eletrónico DW1000 é um módulo transcetor de banda ultra larga UWB da empresa Qorvo, indicado para medições precisas até 12 metros utilizando RTLS e comunicação de dados com velocidade até 6,8 Mbps (Qorvo, 2020).

No desenvolvimento deste projeto, o módulo DW1000 foi utilizado para comunicação entre a *Tag* e a Âncora. A comunicação entre o módulo DW1000 e a placa XIAO ESP32C3 é feita através da interface *Serial Peripheral Interface* SPI, que garante uma transmissão de dados eficiente e fiável (Smith & Lee, 2021; Decawave, 2021). A tabela 3 apresenta as especificações técnicas do módulo DW1000.

Tabela 3 – Especificações técnicas do módulo DW1000, extraído de MikroElektronika (s.d.).

Especificação	Descrição
Faixa de Frequência	3,5 GHz a 6,5 GHz
Taxa de Dados	Até 6,8 Mbps
Precisão de Localização	Até 10 cm
Alcance de Comunicação	Até 300 m (em condições ideais)
Consumo de Energia	Baixo consumo em modo de operação e de sono profundo
Interface de Comunicação	SPI, GPIO
Tensão de Alimentação	3,3 V
Corrente de Saída Digital	10 mA
Temperatura de Operação	-40 °C a +85 °C
Dimensões	28,6 x 25,4 mm (placa UWB Click)
Compatibilidade	Compatível com soquetes mikroBUS™

3.2.2.1. Funcionalidades Relevantes para o Projeto

O módulo DW1000 foi escolhido para este projeto, devido às suas características avançadas: medição da distância com precisão, na ordem dos centímetros, utilizando o método de ToF; suporte de alta taxa de dados até 6,8 Mbps; baixo consumo de energia; amplo alcance em condições favoráveis, e configurações flexíveis que permitem o ajuste de parâmetros e a otimização do desempenho em função das necessidades do projeto (Brown, 2018).

O método ToF baseia-se na medição do tempo que um impulso de nanossegundos leva para viajar entre dois módulos, desde a emissão pelo transmissor até à receção no recetor, e depois retornar ao transmissor. Este tempo é diretamente proporcional à distância entre os módulos. A distância é calculada usando a equação:

$$d = \frac{c.ToF}{2} \quad (1)$$

Onde d representa a distância entre a *Tag* e a Âncora, c a velocidade da luz no ar ($\approx 3 \times 10^8 m/s$) e ToF o tempo do voo do sinal UWB (em segundos). O fator de divisão por 2 é necessário porque o tempo medido inclui a ida e volta do impulso. Este método permite medições precisas, viabilizando

aplicações que requerem alta precisão.

3.2.2.2. Integração com o Sistema

O módulo DW1000 interage com a placa XIAO ESP32C3 utilizando o protocolo SPI padrão, que inclui os pinos MOSI, MISO, SCK e CS, operando à frequência máxima de 20 MHz. Este módulo possui pinos GPIO tanto digitais como analógicos, ver figura 4, que podem ser utilizados para controlar dispositivos (por exemplo, LEDs para sinalização de eventos), bem como interrupções e reinicializações.

A integração do módulo DW1000 neste projeto, em conjunto com a placa XIAO ESP32C3, permitiu a criação de um sistema de medição de distância entre a *TAG* e a Âncora muito eficiente, capaz de comunicar em tempo-real com a aplicação móvel desenvolvida no âmbito deste projeto.

A figura 5 mostra a placa UWB Click utilizada no projeto, que é composta pelo módulo DW1000 e ainda por outros componentes, como resistores, capacitor, além dos indicadores de estado (sinais TX e RX) e sinalização de alimentação de 3,3 V.

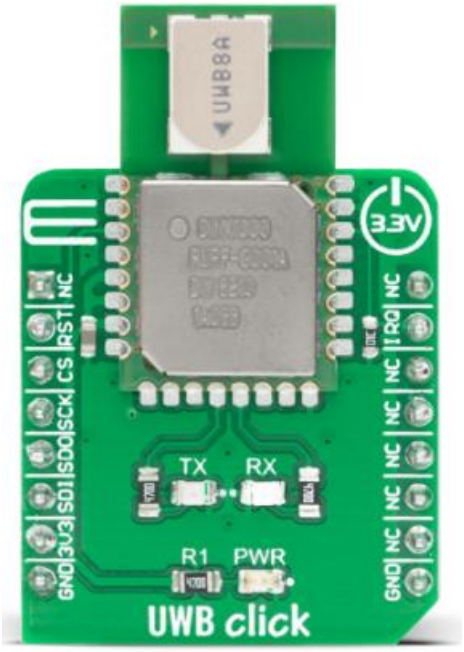

Figura 5 - Imagem do módulo UWB Click, com o transcetor DW1000, extraído de MikroElektronika (s.d.).

3.2.3. Outros Componentes

Além da placa XIAO ESP32C3 e do módulo DW1000, foram utilizados outros componentes eletrónicos, necessários para garantir a comunicação entre os módulos e a alimentação do sistema, incluindo resistor, capacitores e cabos de ligação para a integração com o circuito de alimentação.

3.3. Arquitetura de *Hardware*

Nesta secção é apresentada a arquitetura do sistema de *hardware* desenvolvido, incluindo a estrutura dos componentes principais, a comunicação entre eles e a integração com o sistema de *software*. O principal objetivo é descrever como cada módulo foi projetado para funcionar em conjunto, garantindo o cumprimento dos requisitos do projeto. Tanto a *Tag* quanto a Âncora são constituídas pelos mesmos componentes de *hardware*, diferenciando-se apenas pelo executado em cada uma delas.

3.3.1. Estrutura do sistema de medição

O sistema de medição de distâncias é composto por dois módulos principais: a *Tag*, que é um dispositivo móvel, e a Âncora, que se mantém fixa. Ambos os módulos utilizam a placa XIAO ESP32C3 e o módulo DW1000, para implementar a funcionalidade de medição de distâncias e de comunicação de dados.

Tag: A *Tag* contém um microcontrolador XIAO ESP32C3, responsável pela gestão das comunicações BLE e pela execução do *firmware*. O módulo DW1000, também integrado na *Tag*, permite a comunicação UWB com a Âncora para realizar medições de distância. Após calcular a distância, a *Tag* transmite os resultados via BLE para a aplicação móvel.

Âncora: A Âncora é igual à *Tag* em termos de *hardware*, apesar de ser fixa, servindo como ponto de referência para a medição de distâncias. A comunicação com a *Tag* é estabelecida através de sinais UWB, permitindo que a *Tag* consiga calcular a distância com base no tempo de voo (*ToF*) dos sinais de rádio.

3.3.2. Comunicação entre *Tag* e Âncora

A comunicação entre a *Tag* e a Âncora é realizada utilizando a tecnologia UWB, ideal para medições de alta precisão. A *Tag* é responsável pelo envio dos sinais UWB para a Âncora e quando recebe a resposta da Âncora, calcula a distância com base no tempo que o sinal demora a ir e voltar. A utilização do módulo DW1000 permite uma medição de distância com uma precisão na ordem dos centímetros.

Com a obtenção do ToF é possível calcular a distância entre a *Tag* e a Âncora, medindo-se o tempo que o sinal UWB demora para ser enviado, refletido pela Âncora, e recebido de volta.

A precisão das medições depende da sincronização entre a *Tag* e a Âncora. O módulo DW1000 foi ainda calibrado para garantir que as medições sejam consistentes e precisas.

3.3.3. Integração com a aplicação móvel

Após realizar a medição da distância, a *Tag* utiliza a comunicação BLE para transmitir os dados para a aplicação móvel desenvolvida em Android Studio. Esta transmissão é realizada de forma contínua, permitindo que o utilizador visualize os dados em tempo-real na aplicação.

A *Tag* funciona como um servidor BLE, permitindo que a aplicação conecte a ela como cliente. Os dados de distância são transmitidos em tempo-real e guardados num intervalo de 10 segundos, facilitando a monitorização e visualização do histórico de dados recebidos, caso seja necessário analisar as distâncias medidas entre a *Tag* e a Âncora.

A *Tag* utiliza a comunicação BLE para transmitir os dados para a aplicação móvel, e não a Âncora, devido a vários fatores. Em primeiro lugar, a *Tag* é o dispositivo móvel do sistema, que acompanha o elemento ou o indivíduo cuja distância está a ser medida, enquanto a Âncora é fixa e serve apenas como ponto de referência. Sendo assim, faz mais sentido que seja a *Tag* a responsável pela interface direta com o utilizador através da aplicação, dado que acompanha o objeto rastreado.

3.3.4. Arquitetura de *hardware*

A figura 6 apresenta o diagrama simplificado da arquitetura de *hardware*, ilustrando a disposição dos componentes e a comunicação entre *Tag*, a *Âncora* e a aplicação móvel.

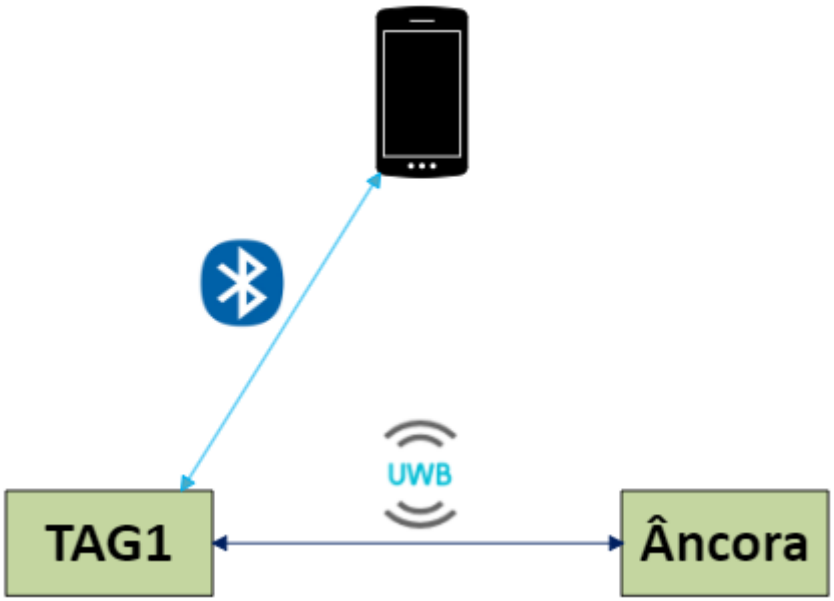

Figura 6 - Arquitetura simplificada do sistema de *hardware*.

3.3.5. Considerações sobre a arquitetura de *Hardware*

A arquitetura de *hardware* foi desenhada para ser modular, permitindo escalabilidade. A utilização de componentes compactos e de baixo consumo energético, como a placa XIAO ESP32C3 e o módulo

DW1000, garante a viabilidade do sistema para aplicações móveis e portáteis. A escolha dos componentes e a disposição dos mesmos foram pensadas para otimizar a precisão das medições e a estabilidade da comunicação.

A integração entre o *hardware* e o *software* é fundamental para o sucesso do projeto. Para melhor compreensão da arquitetura da aplicação e dos serviços associados, recomenda-se a consulta do Capítulo 5, onde é detalhada ao pormenor a interação entre o sistema de *hardware* e a aplicação móvel.

3.4. Calibração da distância entre *Tag* e Âncora

Um dos pontos mais importante do desenvolvimento do sistema de medição, foi a calibração da distância entre a *Tag* e a Âncora. O principal objetivo da calibração é ajustar os parâmetros do sistema, para que a distância medida corresponda à distância real, de forma a minimizar os erros de medição introduzidos por fatores como atraso na antena, condições ambientais, ou eventuais interferências.

3.4.1. Processo de calibração

O atraso da antena é um parâmetro crucial para sistemas da tecnologia UWB que utilizam o módulo DW1000. Serve para compensar o tempo que o sinal leva para percorrer os circuitos internos antes de ser transmitido ou recebido pela antena. Sem esse ajuste, as medições de distância podem apresentar erros.

O processo de calibração envolveu a colocação da *Tag* e da Âncora a uma distância previamente conhecida, medida com o auxílio de uma fita métrica. Essa distância foi comparada com o valor de distância transmitido pelo sistema UWB para a aplicação, onde foi necessário ajustar o valor do atraso da antena, para que os valores coincidissem. Neste sistema, o atraso da antena foi ajustado de acordo com o comportamento observado nos testes para um valor de 16505 unidades que está diretamente relacionado com o modo como o tempo é medido e interpretado pelo sistema de medição UWB.

3.4.2. Método de Calibração

Para garantir a precisão das medições, o processo de calibração foi conduzido da seguinte forma:

1. A Âncora foi posicionada em um local fixo.
2. A *Tag* foi posicionada a diferentes distâncias da âncora, utilizando uma fita métrica para medir a distância real.
3. As medições de distância exibidas pela App desenvolvida foram comparadas com as medições reais, ajustando o valor do atraso da antena até que a diferença entre a distância real e a distância medida fosse minimizada.
4. Foram realizados múltiplos testes, para garantir que o sistema ficasse calibrado corretamente em diferentes distâncias.

Este método assegurou a obtenção de um valor fiável para o atraso da antena, garantido que o sistema funciona de forma consistente em diversos cenários.

Para ajustar o valor do atraso da antena, sabendo que o módulo DW1000 trabalha com uma frequência de 64 GHz, que define o período do *clock* interno, a relação entre a frequência e período é dada pela fórmula:

$$Período\ do\ clock = \frac{1}{Frequência\ (Hz)} \quad (2)$$

Substituindo a frequência de 64 GHz:

$$Período\ do\ clock = \frac{1}{64 \times 10^{9}} = 15{,}625 \times 10^{-12}\ segundos \quad (3)$$

Portanto, cada unidade interna do sistema DW1000 corresponde a $15{,}625 \times 10^{-12}$ segundos, sendo este o menor incremento temporal utilizado pelo módulo.

O atraso interno total da antena foi estimado em 257×10^{-9} segundos. Este valor representa o tempo que o sinal demora a percorrer os circuitos internos do módulo DW1000 e a ser transmitido ou recebido pela antena. Esta estimativa foi obtida com base na documentação técnica do fabricante e nas características especificas do *hardware* utilizado.

Para calcular o atraso da antena em unidade internas, converteu-se o atraso real, medido em nanossegundos, para o período do *clock* do módulo DW1000. A fórmula utilizada foi:

$$Atraso\ da\ Antena = \frac{Atraso\ Real\ (segundos)}{Período\ do\ Clock\ (\frac{segundos}{unidades})} \quad (4)$$

Substituindo os valores:

$$Atraso\ da\ Antena\ (unidades) = \frac{257 \times 10^{-9}}{15{,}625 \times 10^{-12}} \approx 16505\ unidades. \quad (5)$$

Embora o cálculo forneça uma estimativa inicial do atraso da antena, o valor foi ajustado experimentalmente durante os testes. Este processo teve em conta possíveis interferências do ambiente e características especificas do sistema, garantindo que o valor final de 16505 resultasse em medições muito precisas.

3.4.3. Resultados da Calibração

Após o ajuste, o sistema desenvolvido foi testado a diferentes distâncias, e os valores medidos mostraram-se coerentes com a realidade. A figura 7 mostra um exemplo de calibração, onde a distância entre a *Tag* e a Âncora é medida usando uma fita métrica. O valor exibido na aplicação coincide com a distância real, confirmando a precisão do sistema após a calibração.

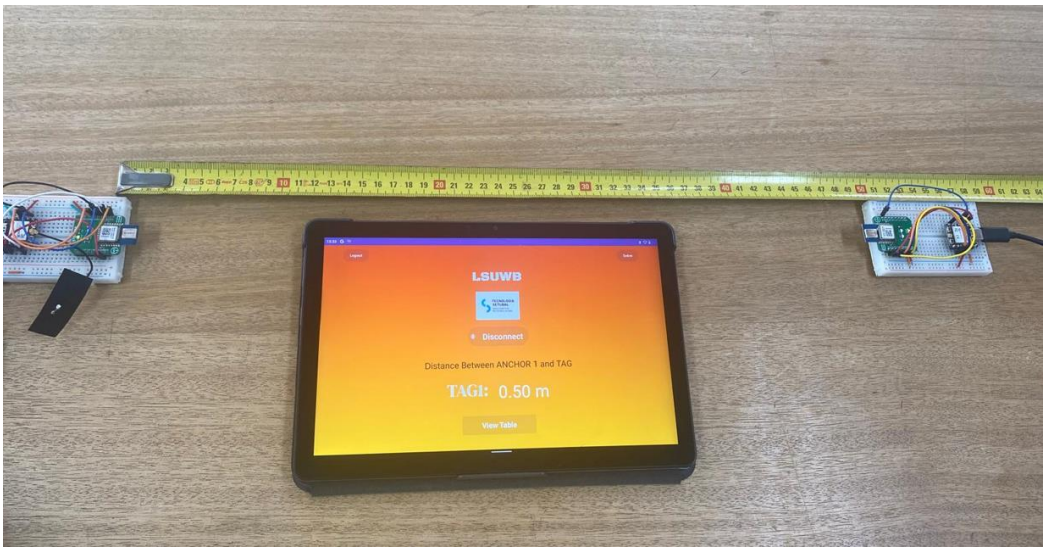

Figura 7 – Precisão da medição após calibração.

3.5. Código Arduíno IDE

Nesta secção, é feita a explicação detalhada das partes mais importantes tanto do código da Âncora como do código da *Tag*.

3.5.1. Código da *Tag*

Na primeira parte do código C++, são definidos os pinos utilizados para a comunicação com o módulo DW1000. Estes pinos são necessários para a ligação SPI e para controlar o módulo. Os pinos configurados têm funções distintas, como por exemplo, o pino de *reset* é utilizado para reiniciar o módulo, o pino de interrupção (PIN_IRQ) para informar eventos, e o pino de seleção SPI (PIN_SS) para controlar a comunicação com o módulo.

3.5.1.1. Definição de Serviço e Características BLE

A *Tag* utiliza a comunicação BLE para comunicar com a aplicação móvel, permitindo que os dados de distância calculados sejam enviados para o cliente BLE. Este serviço é configurado com uma característica que permite a leitura e notificação dos valores, facilitando a comunicação entre a *Tag* e a aplicação.

3.5.1.2. *Callback* do Servidor BLE

O código inclui *callbacks,* que controlam os eventos de ligação e inativação dos clientes BLE. Quando um cliente (como a aplicação móvel) se liga, a *Tag* está pronta para enviar as medições de distância. Caso o cliente se desconecte, o BLE é reiniciado, permitindo novas ligações.

3.5.1.3. Função *writeValueToBytes*

Esta função converte valores de 32 *bits* em *arrays* de *bytes*, necessários para a escrita de dados no módulo DW1000, que funciona com este tipo de formato. É utilizada para ajustar a potência de transmissão.

3.5.1.4. Ajuste da Potência de Transmissão: Função *setTxPower*

A função *setTxPower* ajusta a potência de transmissão do módulo DW1000 de acordo com o canal UWB e a frequência de impulso escolhido. Este ajuste é crítico para otimizar o equilíbrio entre o alcance e o consumo de energia, com a possibilidade de ativar o *smartPower* para reduzir o consumo quando necessário.

3.5.1.5. Inicialização do Sistema

A função *setup ()* inicializa os componentes da *Tag*, como a comunicação UWB e BLE, ajusta a potência de transmissão e prepara o sistema para medir e enviar as distâncias obtidas.

3.5.1.6. Função de Medição de Distância: *newRange()*

A função *newRange()* é responsável por obter a distância calculada pelo módulo DW1000 e transmitir essa informação para a aplicação móvel, via BLE.

3.5.2. Código da Âncora

A Âncora é um dispositivo passivo que responde aos sinais UWB enviados pela *Tag*. Embora a Âncora não calcule diretamente a distância, desempenha um papel importante no processo, participando da comunicação UWB e permitindo que a *Tag* realize a medição da distância.

3.5.2.1. Inicialização e Configuração de Pinos

Tal como na *TAG*, os pinos utilizados para a comunicação SPI e controlo do módulo DW1000 (reset, interrupção e seleção SPI) são configurados da mesma forma. O código de inicialização de pinos na Âncora é idêntico ao da *Tag*.

3.5.2.2. Inicialização do Sistema

Assim como na *Tag*, a função *setup()* na Âncora é responsável por inicializar a comunicação SPI com o módulo DW1000, configurar os parâmetros de operação, ajustar a potência de transmissão, e preparar o módulo para receber os sinais da *Tag*. A principal diferença está no facto de que a Âncora é configurada para responder, enquanto a *Tag* é a responsável por iniciar o processo de medição.

3.5.2.3. Receção de dados UWB

A função *newRange()* é idêntica à da *Tag* no que diz respeito à exibição das medições de distância e da potência do sinal recebido. A Âncora monitoriza o desempenho da comunicação, assegurando que os sinais UWB sejam recebidos corretamente da *Tag*.

3.5.2.4. Função de ajuste de potência

A função *setTxPower* na Âncora é a mesma que na *Tag*, permitindo ajustar a potência de transmissão conforme os parâmetros configurados. Isso é útil para garantir a consistência nos parâmetros de comunicação entre a *Tag* e a *Âncora*, especialmente em cenários de maior complexidade.

O sistema está configurado no canal 5, com frequência de pulso de 64 MHz e *Smart Power* ativado, com o valor de 0x25456585 que corresponde a uma potência de transmissão de aproximadamente -10 dBm, O *Smart Power* reduz automaticamente a potência, ajustando-a para otimizar o consumo de energia e garantir a conformidade com os limites regulamentares.

Neste capítulo, foi detalhado o desenvolvimento do *hardware* para o sistema de medição de distâncias utilizando a tecnologia UWB e a comunicação BLE. A escolha dos componentes principais, como a placa XIAO ESP32C3 e o módulo DW1000 foi fundamentada pelas suas capacidades técnicas, baixo preço e consumo energético e elevada precisão na medição de distâncias. A implementação do código para ambos os módulos, bem como as configurações de comunicação e transmissão de dados, garantem a medição precisa e a comunicação em tempo-real. Com os testes e ajustes realizados, este *hardware* constitui uma base sólida para o sistema de localização desenvolvido, pronto para ser integrado com a aplicação descrita no capítulo 5.

Capítulo 4
Desenho das placas em KiCad

4.1. Introdução ao KiCad

Este capítulo detalha o processo de desenho das Placas de Circuito Impresso (PCBs) para o sistema de medição desenvolvido, utilizando a ferramenta *freeware* KiCad. O KiCad é um *software open-source* para a criação de esquemas elétricos e *layout* de PCBs. Ao longo deste capítulo, serão exploradas as etapas de conceção do esquema eletrónico, o *layout* da PCB, e as considerações específicas para a integração dos módulos XIAO ESP32C3.

4.2. Desenho do esquema no Schematic editor

4.2.1. Configuração Inicial

O projeto foi iniciado no KiCad com a criação de um novo ambiente de trabalho, onde as bibliotecas de componentes foram configuradas de acordo com as necessidades específicas do desenho. Componentes-chave como a placa XIAO ESP32C3 e o modulo DW1000 foram adicionados e destacados nas bibliotecas, garantindo que todos os elementos essenciais estivessem disponíveis para o desenvolvimento do esquemático.

4.2.2. Desenho do esquema elétrico

Com as bibliotecas configuradas, iniciou-se o desenho com a integração dos componentes no *Schematic Editor*. A seleção cuidadosa dos componentes e sua integração no esquema garantiu a representação precisa do circuito proposto. As ligações elétricas entre os componentes foram realizadas seguindo a lógica do circuito, conforme ilustrado na figura 8. Os componentes foram identificados de forma a facilitar futuras intervenções no processo de desenho.

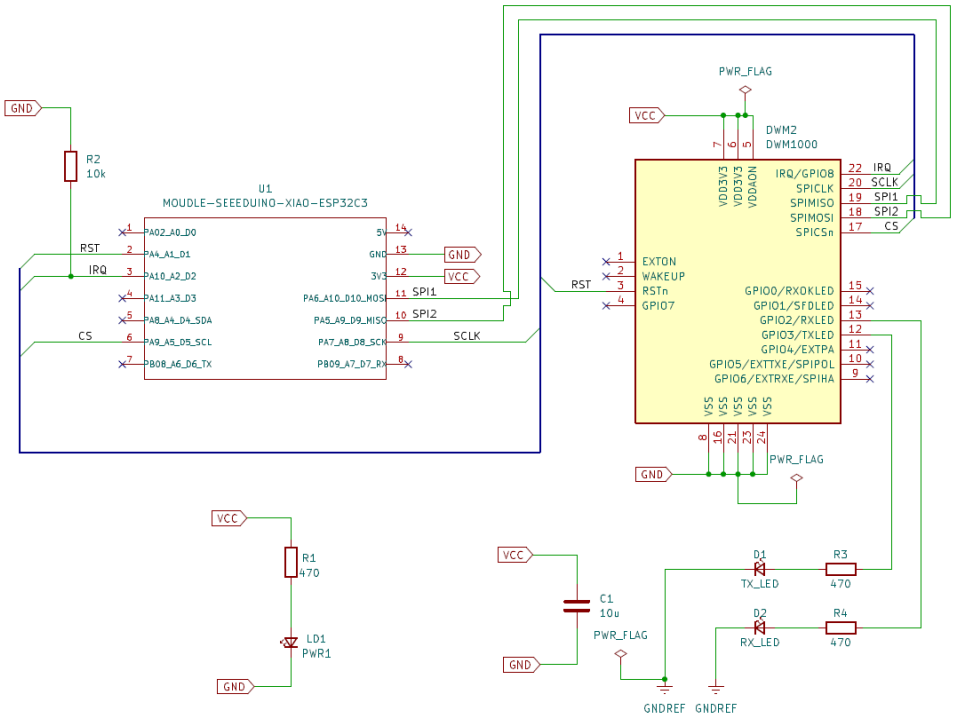

Figura 8 - Esquema elétrico da *Tag*[1] desenvolvido no Schematic Editor do KiCad.

4.2.2.1. Verificação do esquema

Após a conclusão do desenho do esquema elétrico, foi realizada uma verificação de erros utilizando o *Electrical Rules Checker* (ERC) do KiCad. Esta ferramenta é fundamental para identificar possíveis erros ou inconsistências no esquemático antes de avançar para a etapa de *layout* da PCB.

O ERC é projetado para detetar problemas elétricos no circuito, como ligações incorretas, pinos não conectados, ou conflitos de tipo de pino. No processo de verificação do projeto, o ERC não identificou erros críticos, o que indica que o esquema foi desenhado corretamente conforme os requisitos do projeto e as especificações dos fabricantes.

Esta etapa de verificação é essencial para assegurar que o circuito projetado é funcional e isento de problemas que poderiam comprometer a operação da PCB em fase de fabrico ou durante a sua utilização final.

4.3. Desenho do esquema no PCB editor

Com o esquema elétrico concluído e verificado, o próximo passo foi a importação do esquema para o PCB Editor do KiCad, onde o *layout* físico da placa de circuito impresso da *Tag* foi desenhado com duas camadas e foi criado o plano de massa para minimizar interferências eletromagnéticas. Este

[1] A *Tag* e a Âncora são constituídos pelos mesmos componentes de *hardware.*

processo envolveu a organização dos componentes, o roteamento das trilhas e a realização de verificações para assegurar que o desenho atende a todas as regras de fabricação.

4.3.1. Importação do Esquema para o PCB Editor

Após a verificação do esquema, os componentes foram mapeados e importados para o PCB Editor. Durante essa fase, foi necessário definir o tamanho e as camadas da PCB de duas camadas para acomodar todos os componentes de forma eficiente. A figura 9 mostra o *layout* final da PCB, onde os componentes foram posicionados estrategicamente para otimizar o espaço, e sendo ainda criado o plano de massa, que é recomendável para minimizar interferências eletromagnéticas.

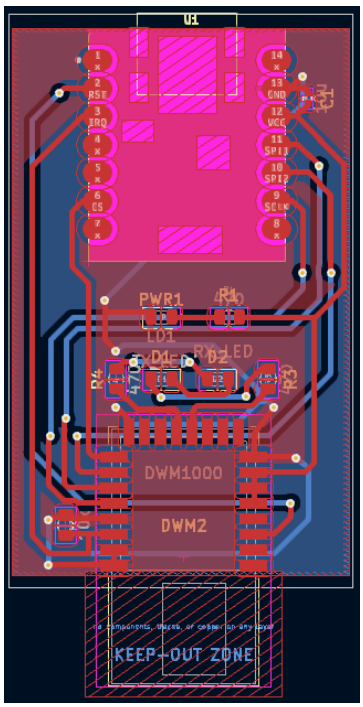

Figura 9 – *Layout* da PCB da *Tag*[2] no PCB Editor do KiCad.

4.3.2. Organização dos componentes na PCB

Os componentes eletrónicos foram organizados com o objetivo de reduzir o comprimento das pistas e evitar problemas de interferência eletromagnética. A rotação e o ajuste dos componentes foram realizados, de modo a garantir um *layout* compacto e funcional. A figura 10 apresenta a visualização 3D da PCB da *Tag*, que permite observar a disposição física dos componentes e a estética final do desenho.

[2] A *Tag* e a Âncora são constituídos pelos mesmos componentes de *hardware.*

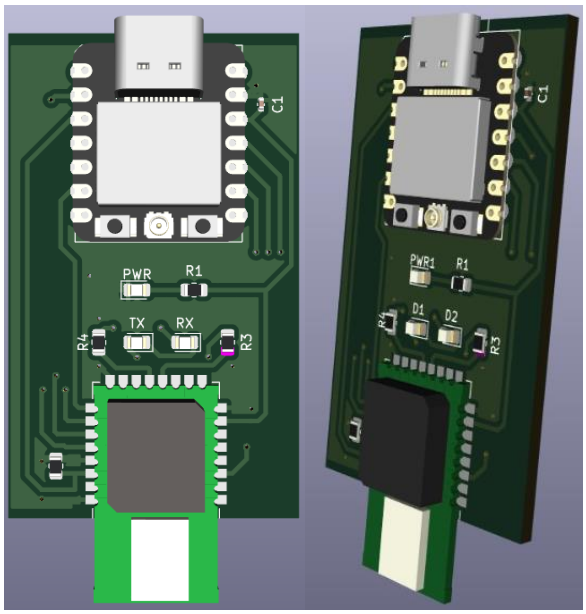

Figura 10 – Visualização 3D da PCB no KiCad.

4.3.3. Roteamento das pistas

O roteamento das pistas foi realizado com base nas especificações elétricas e nos requisitos de sinal dos componentes principais, como os módulos XIAO ESP32C3 e DW1000. A largura das pistas e o espaçamento entre elas foram cuidadosamente escolhidos para garantir a integridade do sinal e a capacidade de corrente das ligações. Além disso, foi implementado um plano de massa para melhorar o desempenho elétrico da placa, especialmente em termos de imunidade ao ruído e interferências eletromagnéticas.

4.3.4. Verificação e ajustes do *layout*

Após o roteamento das pistas, o *layout* foi submetido ao *Design Rules Checker* (DRC) do KiCad, uma ferramenta essencial para identificar possíveis problemas que possam comprometer a fabricação ou o funcionamento da PCB. O DRC não identificou erros críticos.

4.3.5. Lista de material

Na tabela 4, encontra-se a lista com todos os componentes utilizados no desenvolvimento do projeto.

Tabela 4 – Lista de material de todos os componentes usados no projeto.

Id	***Designador***	***Dimensões***	**Quantidade *(un)***	***Designação***
1	C1	C_0402_1005Metric	1	10µF capacitor
2	R1,R3,R4	R_0805_2012Metric	3	470 ohm resistor
3	U1	MOUDLE14P-SMD-2.54-21X17.8MM	1	MOUDLE-SEEEDUINO-XIAO-ESP32C3
4	D1	LED_0805_2012Metric	1	TX_LED
5	R2	R_0805_2012Metric	1	10k ohm resistor
6	DWM2	DWM1000	1	DWM1000 module
7	PWR1	LED_0805_2012Metric	1	Led power
8	D2	LED_0805_2012Metric	1	RX_LED

Neste capítulo, foi apresentado o processo completo de desenho das Placas de Circuito Impresso (PCBs) da *Tag* utilizando a ferramenta KiCad, desde a configuração inicial do projeto até à preparação final para a fabricação. Foi destacada a importância de uma abordagem estruturada, começando pelo desenho do esquema elétrico no *Schematic Editor*, onde as ligações entre os módulos principais, como o XIAO ESP32C3 e o DW1000, foram cuidadosamente planeadas e verificadas.

O processo de desenho da PCB no *PCB Editor* incluiu a importação do esquemático, a organização dos componentes, e o roteamento das pistas, com ênfase na integridade dos sinais e na eficiência do *layout*. A verificação através do *Design Rules Checker* (DRC) assegurou que o desenho estava de acordo com as regras de fabricação, com os *warnings* identificados sendo considerados não críticos para a produção.

Além disso, a integração dos módulos XIAO ESP32C3 e DW1000 exigiu considerações específicas, tanto em termos de desenho elétrico quanto de *layout* físico, garantindo que os requisitos de desempenho fossem plenamente atendidos. As revisões e ajustes finais, baseados em *feedback* e validações, reforçaram a robustez do desenho obtido.

Demonstrou-se não apenas a aplicação eficaz do KiCad como ferramenta de desenho de *hardware*, mas também a importância de existir um processo de desenvolvimento iterativo, onde cada etapa foi validada e otimizada para garantir a qualidade e funcionalidade da PCB final. O sucesso na criação de uma PCB da *Tag*, que atende aos requisitos técnicos e estéticos estabelecidos, evidência a competência técnica aplicada ao longo do projeto e prepara o terreno para as fases subsequentes de desenvolvimento e testes.

Capítulo 5
Desenvolvimento da App

5.1. Introdução ao desenvolvimento da aplicação

Neste capítulo, é apresentada a aplicação móvel desenvolvida para interagir com o sistema de localização UWB, permitindo ao utilizador visualizar, em tempo-real, a distância entre uma *Tag* e uma Âncora.

A App, desenvolvida na ferramenta Android Studio, comunica com a *Tag* utilizando a tecnologia BLE integrada no módulo XIAO ESP32C3. O projeto da aplicação foca-se na simplicidade de uso, gestão eficaz de permissões e integração com o *Firebase* para autenticação e armazenamento dos dados. Este capítulo descreve em detalhe o processo de desenvolvimento, desde a conceção da interface até à implementação das funcionalidades essenciais de comunicação.

5.2. Arquitetura da aplicação

A aplicação é construída sobre uma arquitetura modular, que inclui diversas atividades e serviços essenciais para o funcionamento do sistema de medição UWB. Cada componente foi projetado para executar tarefas específicas dentro do fluxo de operação geral, garantindo que o sistema seja tanto eficiente quanto escalável. O diagrama da arquitetura, ilustrado na figura 11, mostra a organização dos principais componentes, incluindo a comunicação entre a *Tag* e a Âncora via UWB, e a transmissão dos dados para a App, via *Bluetooth*.

Na ferramenta Andoid Studio existem duas opções de linguagem de programação (Java e Kotlin) para desenvolvimento da aplicação. No entanto escolheu-se o Java como linguagem de programação principal, devido à sua robustez e amplo suporte para desenvolvimento Android. A aplicação conta com uma arquitetura que inclui várias atividades e serviços, designadamente para lidar com a autenticação de utilizadores, a gestão da sessão, e a apresentação dos dados.

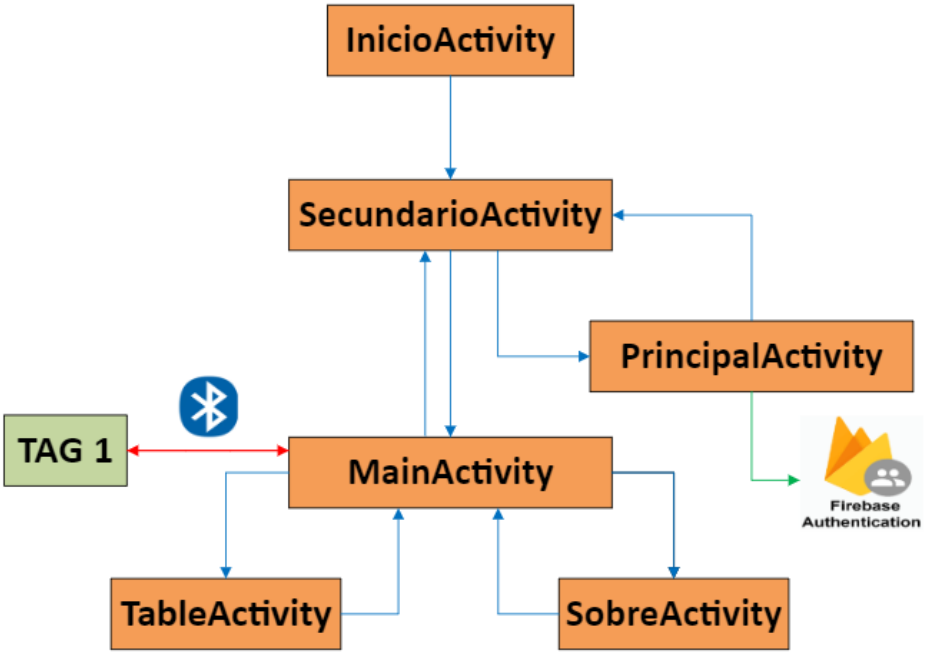

Figura 11 – Arquitetura da aplicação móvel.

A *InicioActivity* é a tela inicial, que exibe uma mensagem de boas-vindas, fazendo a transição automática para a *SecundarioActivity* após 3 segundos. Estando na *SecundarioActivity*, o utilizador pode iniciar a secção ou fazer registo do utilizador. No caso de ser a primeira utilização e de necessitar de fazer o registo é direcionado para a tela da *PrincipalActivity* para registar novo utilizador. Quando o utilizador faz o registo é direcionada para a tela da *MainActivity*, permitindo-lhe ligar-se ao dispositivo UWB (TAG1), iniciar e parar a medição de distâncias, e visualizar o histórico de medições numa tabela.

5.3. Desenvolvimento de interfaces

O desenvolvimento das interfaces foi criado para que o utilizador tenha uma experiência de utilização intuitiva e acessível. Utilizando a ferramenta Android Studio, foram criadas interfaces visuais claras e simples, que facilitam a navegação e a interação com as funcionalidades da App. As telas foram desenhadas para serem autoexplicativas, com botões e ícones grandes, de modo a facilitar o acesso em dispositivos móveis.

5.3.1. Atividades (*Activities*)*:*

A *InicioActivity* é a tela inicial, que exibe uma mensagem de boas-vindas, fazendo a transição automática para a atividade principal, após três segundos. Esta atividade é a primeira janela que surge quando o utilizador abre a aplicação. A captura de ecrã da interface da *InicioActivity* é apresentada na figura 12.

Figura 12 – *InicioActivity*.

A atividade *MainActivity* é onde o utilizador tem maior interação com a aplicação, permitindo-lhe ligar-se ao dispositivo UWB (*TAG1*), iniciar e parar a medição de distâncias, e visualizar o histórico de medições numa tabela. Na figura 13 é apresentada uma captura de ecrã da *MainActivity*.

Figura 13 – Tela *MainActivity*.

As atividades *SecundarioActivity* e *PrincipalActivity* compõem o conjunto de atividades responsáveis pela gestão da autenticação e do registo de novos utilizadores, utilizando o Firebase como *back-end* para o armazenamento de dados dos utilizadores. A figura 14 ilustra as capturas de ecrã da *SecundarioActivity* e da *PrincipalActivity*.

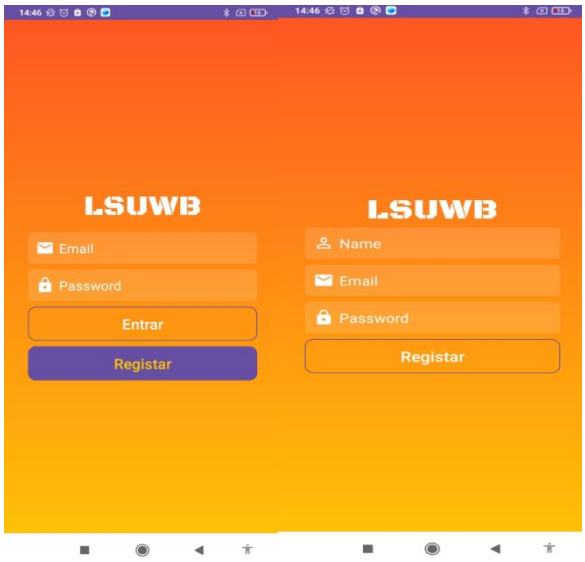

Figura 14 – Tela *SecundarioActivity* e *PrincipalActivity*.

A *SobreActivity* fornece informações sobre a aplicação. A figura 15 apresenta a *SobreActivity*.

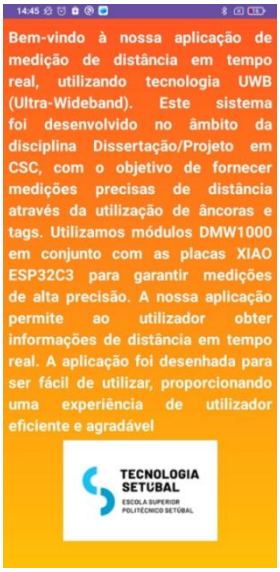

Figura 15 – Tela *SobreActivity*.

A *TableActivity* exibe o histórico detalhado de medições realizados pelo sistema numa tabela, permitindo ao utilizador visualizar as medições numa janela temporal de 10 em 10 segundos, sendo possível verificar o registo da data, da hora e da distância, de forma simples e intuitiva. O valor da janela temporal foi escolhido como um valor de equilíbrio para registo de mudanças significativas sem sobrecarregar o sistema. Contudo, este intervalo pode ser ajustado para valores menores ou maiores, conforme as necessidades do utilizador ou do contexto de aplicação. A figura 16 apresenta uma captura de ecrã da *TableActivity*.

A organização em formato de tabela tem como objetivo fornecer uma visão clara e estruturada, facilitando a análise e identificação de padrões ou anomalias nas medições.

Figura 16 – Tela *TableActivity*.

5.4. Gestão de permissões e conectividade *Bluetooth*

No desenvolvimento da aplicação foram criadas várias permissões importantes, destacando-se o acesso ao *Bluetooth* e à localização do dispositivo. Na secção 5.8 é possível verificar no código as permissões que foram definidas. A comunicação *Bluetooth* é essencial para a comunicação com os módulos UWB. A App concebida utiliza APIs do sistema operativo Android para encontrar e ligar a dispositivos BLE de forma eficiente, garantindo que as medições de distância são recebidas em tempo-real.

5.5. Autenticação e gestão de utilizadores

A aplicação utiliza uma combinação de serviços e de *Broadcast Receivers*, para gerir eficazmente várias funcionalidades essenciais, proporcionando uma experiência de utilizador fluida e eficiente. O *Bluetooth Management* desempenha um papel fundamental na comunicação com dispositivos UWB, sendo responsável pela procura, estabelecimento de ligação e manutenção da comunicação com esses dispositivos. Este serviço também processa os dados de distância recebidos, assegurando que as informações são corretamente interpretadas e transmitidas para outras partes da aplicação.

O *Range Update Receiver* é um *BroadcastReceiver* específico, que se encarrega de escutar atualizações de distância emitidas por outros componentes da aplicação. Sempre que uma nova leitura de distância é recebida ou uma atualização ocorre, este recetor intervém para garantir que a interface do utilizador é atualizada em tempo-real, refletindo as medições mais recentes. Esta funcionalidade é crucial para aplicações que dependem de dados em tempo-real, como as que monitorizam a proximidade de objetos ou dispositivos.

Além dos aspetos técnicos relacionados com a gestão das ligações e dados, a integração com o *Firebase* acrescenta uma camada robusta de funcionalidades relacionadas à autenticação e à gestão de dados. O *Firebase Auth* é utilizado para gerir o processo de *login* e de registo de utilizadores, oferecendo uma solução segura e eficiente para a criação e gestão de contas de utilizador. A aplicação permite que os utilizadores se registem e iniciem sessão utilizando os seus endereços de email e senhas, garantindo que apenas utilizadores autenticados tenham acesso às funcionalidades principais.

O *Firebase Realtime Database* complementa esta integração ao fornecer uma solução escalável e em tempo-real para o armazenamento e recuperação dos dados dos utilizadores. Nesta aplicação, o banco de dados é utilizado para armazenar informações críticas, como o histórico de medições de distância e as preferências do utilizador. A arquitetura em tempo-real do *Firebase* permite que os dados sejam sincronizados instantaneamente entre os dispositivos ligados e o servidor, garantindo que as informações estejam sempre atualizadas e disponíveis para os utilizadores, independentemente do dispositivo que estejam a utilizar.

O *Firebase* também facilita a gestão centralizada de dados, permitindo que as configurações e preferências do utilizador sejam armazenadas de forma segura e recuperadas conforme necessário. Isto não apenas melhora a experiência do utilizador, mas também simplifica a manutenção e a atualização da aplicação, uma vez que os dados podem ser geridos e atualizados a partir de um único ponto centralizado.

Por fim, a aplicação permite uma personalização fácil e segura das contas de utilizador, com o *Firebase Auth*, gerindo operações como redefinição de senha, verificação de email e até autenticação de dois fatores, se necessário. A figura 17 ilustra o processo de registo de novos utilizadores no *Firebase*, destacando como o fluxo de autenticação foi integrado, para proporcionar uma experiência de utilizador intuitiva e segura.

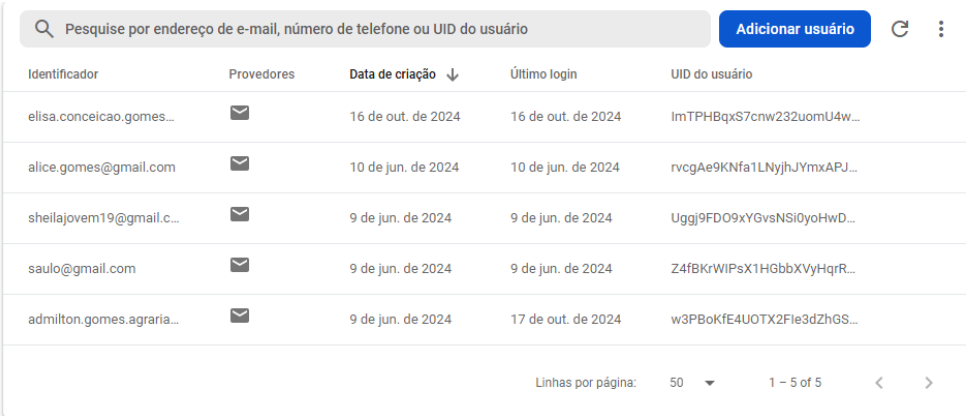

Figura 17 – Autenticação *Firebase*.

5.6. Testes e validação

A realização de testes e de validações é uma etapa crucial no desenvolvimento de *software*, especialmente em aplicações móveis, onde a diversidade de dispositivos e configurações pode introduzir uma vasta gama de comportamentos e de potenciais problemas. Nesta secção serão discutidos os diferentes tipos de testes realizados, as metodologias adotadas e as ferramentas utilizadas, para garantir que a aplicação cumpre os requisitos funcionais e não funcionais.

5.6.1. Testes unitários

Os testes unitários consistem na validação de pequenas unidades de código, como funções ou métodos, garantindo que cada componente individual funciona conforme esperado. No contexto desta aplicação, foram desenvolvidos testes unitários para verificar a correta funcionalidade dos métodos críticos, como os de processamento de dados de distância, cálculo de medições e manipulação de dados dos utilizadores. Ferramentas como JUnit e Mockito foram utilizadas para criar cenários de teste, que simulam diferentes condições e verificam se as saídas correspondem aos resultados esperados.

5.6.2. Testes de integração

Os testes de integração foram conduzidos para assegurar que os diferentes módulos da aplicação funcionam harmoniosamente quando combinados. Por exemplo, a integração entre o módulo de *Bluetooth Management* e o *Range Update Receiver* foi testada, para garantir que as leituras de distância são corretamente capturadas e refletidas em tempo-real na interface do utilizador. Além disso, a integração com o Firebase foi exaustivamente testada para verificar a consistência dos dados entre o cliente e o servidor, especialmente durante operações críticas como o *login*, registo de utilizadores e sincronização dos dados.

5.6.3. Testes funcionais

Os testes funcionais avaliam se a aplicação cumpre todos os requisitos funcionais especificados. Estes testes envolveram a execução de casos de estudo típicos, como o processo de ligação a um dispositivo UWB, a medição de distâncias, a visualização do histórico de medições, e a gestão de contas de utilizador. Foram simulados diferentes cenários, como a perda de conectividade *Bluetooth* e a recuperação da ligação, para garantir que a aplicação responde adequadamente a essas situações. Ferramentas de automação de testes, como Expresso, foram utilizadas para automatizar parte destes testes em diferentes dispositivos e em várias versões do Android.

5.6.4. Testes de usabilidade

A usabilidade é um aspeto fundamental para o sucesso de qualquer aplicação móvel. Foram realizados testes de usabilidade para avaliar a experiência do utilizador ao interagir com a aplicação. Utilizadores reais foram convidados a testar a aplicação em diferentes cenários de uso, dando *feedback* sobre a facilidade de navegação, a clareza das interfaces e a eficiência das operações. Este *feedback* foi crucial para identificar áreas de melhoria e ajustar o projeto da interface e o fluxo de trabalho da aplicação.

5.6.5. Testes de desempenho

Os testes de desempenho focaram-se em avaliar a resposta e a eficiência da aplicação sob diferentes condições de utilização. Foram realizadas simulações para testar o comportamento da aplicação quando submetida a um elevado volume de dados ou a ligações *Bluetooth* simultâneas. Além disso, o desempenho da integração com o *Firebase* foi avaliado, especialmente em cenários de sincronização em tempo-real de grandes volumes de dados. Ferramentas como Android Profiler foram utilizadas para monitorizar a utilização de recursos, como CPU, memória e bateria, identificando potenciais constrangimentos de desempenho.

A tabela 5 apresenta um resumo dos principais cenários testados e os respetivos resultados.

Tabela 5 – Resultados dos testes de desempenho da aplicação.

Cenário de Teste	Objetivo	Métricas Monitorizadas	Resultados Observados
Elevado volume de dados	Avaliar a sincronização de 10.000 registos por minuto no Firebase	CPU (%), Latência (ms), Consistência	CPU média de 65%; latência de 250 ms; dados sincronizados corretamente sem perdas ou falhas.
Ligações Bluetooth simultâneas	Testar a estabilidade com 5 dispositivos ligados simultaneamente	Consumo de CPU (%), Estabilidade	Estabilidade mantida; CPU subiu para 75%; sem falhas de ligação ou quedas na aplicação.
Sincronização em tempo real com o Firebase	Validar a atualização contínua de 1.000 registos em menos de 10 s	Latência (ms), Consistência dos Dados	Sincronização concluída em 8 s; latência média de 150 ms; sem perdas ou duplicações de dados.
Consumo de recursos	Medir o impacto de 1 hora de operação contínua (BLE + Firebase)	Bateria (%), Temperatura do dispositivo	Consumo de bateria foi de 20% em 1 hora; temperatura do dispositivo estabilizada em 37°C, sem impacto no desempenho.
Sobrecarga do sistema	Enviar 20.000 registos/min para simular uma carga extrema	CPU (%), Latência (ms), Estabilidade	CPU atingiu 90%; latência aumentou para 500 ms; atrasos pontuais na sincronização observados, mas sem falhas críticas no sistema.

5.6.6. Testes de segurança

A segurança é uma prioridade, especialmente quando se lida com dados de utilizadores e informações sensíveis. Foram realizados testes de segurança, para garantir que a aplicação protege adequadamente os dados dos utilizadores e resiste a tentativas de acesso não autorizado. O fluxo de autenticação gerido pelo *Firebase* foi testado, para garantir a segurança do *login* e do armazenamento dos dados. Além disso, foram conduzidos testes para avaliar a resistência da aplicação a vulnerabilidades comuns, como modificações de código e manipulação de sessões.

5.6.7. Validação final

Após a realização dos testes mencionados, foi realizada uma validação final da App, para verificar se todas as funcionalidades implementadas estavam em conformidade com os requisitos estabelecidos no início do projeto da App. Esta validação envolveu a execução de todos os casos de uso principais, a verificação da integridade dos dados armazenados e a avaliação da aplicação em diferentes dispositivos e ambientes. A validação final confirmou que a aplicação não só cumpre os seus objetivos iniciais, como também oferece uma experiência de utilizador robusta e segura.

5.7. Desafios e soluções

Durante o desenvolvimento da aplicação móvel, diversos desafios técnicos surgiram, especialmente na implementação das funcionalidades da ligação *Bluetooth* e integração com o Firebase. Nesta secção abordamos os principais desafios encontrados e as soluções implementadas para garantir que a aplicação atingisse os objetivos pretendidos.

5.7.1. Gestão de ligação *Bluetooth*

Um dos maiores desafios foi garantir uma ligação *Bluetooth* estável e confiável com dispositivos UWB, como o *ESP32_Ranging*. O código da *MainActivity* revela o esforço em gerir a conectividade, desde a busca por dispositivos até à manutenção da ligação ativa. Foi necessário implementar verificações rigorosas de permissões e a capacidade de ligação automática em caso de falhas de comunicação. A solução envolveu o recurso ao *BluetoothGattCallback*, para monitorizar o estado da ligação e garantir que, em caso de desconexão, a aplicação tentasse reconectar ou notificasse o utilizador de maneira adequada. Além disso, a implementação de *handlers* para gerir a frequência de recolha de dados garantiu que as leituras de distância fossem feitas de forma contínua e eficiente.

5.7.2. Sincronização de dados em tempo-real

A sincronização das leituras de distância em tempo-real foi um desafio significativo. O código faz uso de *Broadcasts* para atualizar a interface do utilizador sempre que uma nova leitura é registada. Essa abordagem garante que a interface permaneça responsiva e atualizada. Contudo, a sincronização dos dados com o *Firebase*, especialmente o histórico de medições, exigiu a otimização das operações de rede para minimizar o impacto no desempenho da aplicação. O uso de *Handler* para executar tarefas em segundo plano foi crucial para garantir que a aplicação permaneça responsiva enquanto processava e transmitia dados.

5.7.3. Gestão de permissões e segurança

Gerir as permissões necessárias para o *Bluetooth* e o acesso à localização foi um desafio crítico. O código contém verificações explícitas de permissões para garantir que a aplicação só tenta realizar operações *Bluetooth* quando todas as permissões necessárias foram concedidas. Isso foi especialmente importante para evitar falhas de segurança e garantir conformidade com as políticas de permissões do sistema operativo Android (Yang et al., 2020). A solução foi implementar um sistema que solicita as permissões de forma incremental e somente quando necessário, proporcionando uma melhor experiência ao utilizador e minimizando as chances de o utilizador recusar permissões críticas.

5.7.4. Desconexões e reconexões automáticas

Outro desafio enfrentado foi lidar com as desconexões inesperadas de dispositivos *Bluetooth*. O código implementa uma lógica de desconexão e reconexão, que garante que a aplicação tente restabelecer a ligação automaticamente quando possível, e realiza a limpeza adequada dos recursos quando a desconexão é permanente. Essa abordagem minimizou a frustração do utilizador ao diminuir a necessidade de intervenção manual para reconectar o dispositivo.

5.7.5. Testes em ambientes diversos

Testar a aplicação em diferentes dispositivos e versões do Android apresentou desafios, especialmente em termos de compatibilidade de *Bluetooth*. A necessidade de assegurar que o código funcionasse em uma ampla gama de dispositivos levou à realização de testes extensivos, tanto em dispositivos físicos como em simuladores. A solução consistiu em ajustes refinados no código para lidar com as peculiaridades de diferentes versões do Android e *stacks Bluetooth*, garantindo uma experiência de utilizador consistente.

5.7.6. Integração com o Firebase

A integração com o Firebase, especialmente para autenticação e armazenamento de dados, apresentou desafios no que diz respeito à segurança e à gestão eficiente dos dados de utilizadores. A solução envolveu a configuração de regras de segurança personalizadas no Firebase, garantindo que apenas utilizadores autenticados pudessem acesar seus dados, além de implementar práticas recomendadas para a segurança e a eficiência da comunicação entre a aplicação e o servidor.

5.8. Código

5.8.1. AndroidManifest.xml

O *AndroidManifest.xml* define as permissões, configurações gerais e componentes essenciais que garantem o bom funcionamento da aplicação no sistema Android.

5.8.1.1. Declaração de permissões

O elemento *"<manifest>"* define o namespace XML (figura 18), necessário para os atributos específicos do Android e estabelece o espaço de nomes do pacote da aplicação, que é utilizado internamente pelo Android, para identificar de forma única a aplicação. Este elemento é essencial para a correta configuração do ambiente de execução da aplicação, assegurando que todas as permissões e atributos necessários sejam declarados e reconhecidos pelo sistema operativo Android.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.agodist">
```

Figura 18 – Estrutura inicial do *manifest.*

O elemento *"<uses-permission>"* (figura 19) especifica as permissões que a aplicação requer para aceder ao *hardware* ou aos dados no dispositivo. A permissão *BLUETOOTH* permite que a aplicação realize tarefas relacionadas com a comunicação *Bluetooth*, como localizar e ligar dispositivos. A permissão *BLUETOOTH_ADMIN* autoriza a realização de tarefas administrativas de *Bluetooth*, como ativar ou desativar este tipo de comunicação. Já a permissão *ACCESS_FINE_LOCATION* é necessária para aceder à localização precisa, o que é fundamental para a funcionalidade de localização UWB.

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

Figura 19 – Declaração de permissões para ligação *Bluetooth.*

5.8.1.2. Configuração da aplicação

O elemento *"<application>"* (figura 20) define os atributos globais da aplicação. O atributo *allowBackup* habilita ou desabilita a capacidade de *backup* e a recuperação dos dados da aplicação, permitindo que os utilizadores mantenham os seus dados ao trocar de dispositivo. Os atributos *icon* e *roundIcon* definem os ícones utilizados na janela de início e em outros locais do sistema operativo. O atributo *label* indica o nome da aplicação, conforme apresentado no dispositivo. O atributo *supportsRtl* especifica se a aplicação suporta *layouts* da direita para a esquerda, enquanto o atributo *theme* define o tema visual da aplicação.

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.AGODIST">
```

Figura 20 – Configuração do AndroidManifest.xml com ícones, tema e RTL.

5.8.1.3. Declaração de atividades

O elemento *"<activity>"* apresentado na figura 21, define cada uma das telas da aplicação. O atributo *exported* controla se outras aplicações podem iniciar essa atividade. A *InicioActivity* encontra-se marcada com *exported="true"* e contém um filtro de intenção que a designa como o ponto de entrada da aplicação.

```
<activity
    android:name=".SecundarioActivity"
    android:exported="false" />
<activity
    android:name=".PrincipalActivity"
    android:exported="false" />
<activity
    android:name=".SobreActivity"
    android:exported="false" />
<activity
    android:name=".MainActivity"
    android:exported="false" />
<activity
    android:name=".InicioActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
<activity android:name=".TableActivity" />
```

Figura 21 – Configuração de atividades.

O elemento *"<meta-data>"* apresentado na figura 22, especifica metadados adicionais para a aplicação. Neste caso, é utilizado para indicar as fontes pré-carregadas, o que melhora o desempenho da aplicação, ao reduzir o tempo de carregamento das fontes frequentemente utilizadas.

```
  <meta-data
        android:name="preloaded_fonts"
        android:resource="@array/preloaded_fonts" />
</application>
```

Figura 22 – Configuração de fontes pré-carregadas.

No anexo I encontra-se o código completo do *AndroidManifest.xml*, conforme explicado acima.

5.8.2. Layouts da aplicação

A aplicação é composta por seis *layouts*, que definem a forma como certos elementos visuais, como botões, textos e imagens são apresentados, e dispostos na tela, influenciando diretamente a interatividade da aplicação. Assim, os *layouts* foram desenhados para suportar funcionalidades essenciais, como a medição de distâncias, a autenticação de utilizadores e a visualização de dados históricos, proporcionando uma interface com o utilizador humano simples e acessível.

A seguir, serão descritos detalhadamente os principais *layouts* utilizados na aplicação, incluindo as suas funções e os componentes que os constituem. A explicação é acompanhada pelo código XML correspondente, permitindo uma compreensão clara e direta da sua implementação.

5.8.2.1. Activity_inicio.xml

A tela definida pelo ficheiro *Activity_inicio.xml* tem como objetivo funcionar como a página de boas-vindas da aplicação. Esta tela apresenta o desenvolvedor e o logotipo do IPS, e realiza uma transição suave para a tela principal após alguns segundos, criando uma primeira impressão positiva para o utilizador.

Componentes principais:

- ***TextView:***
- Este componente é utilizado para exibir uma mensagem de boas-vindas, introduzindo-o ao ambiente da aplicação. Na figura 23 é apresentada a configuração da *TextView* em XML.

```
<TextView
    android:id="@+id/textView5"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:text="Bem-Vindo"
    android:textSize="40dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.0" />
```

Figura 23 – Configuração *TextView* em XML.

- ***ImageView:***
- O *ImageView* é apresentado na figura 24, sendo responsável por mostrar o logotipo da aplicação (logotipo do IPS), reforçando a identidade visual da aplicação e a sua associação institucional.

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="199dp"
    android:layout_height="116dp"
    android:layout_gravity="center"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.478"
    app:layout_constraintStart_toStartOf="@+id/textView5"
    app:layout_constraintTop_toTopOf="@+id/textView5"
    app:layout_constraintVertical_bias="0.026"
    app:srcCompat="@drawable/bag_ips" />
```

Figura 24 – Configuração do *ImageView*.

- ***ProgressBar:***
- A *ProgressBar* apresentado na figura 25, proporciona *feedback* visual ao utilizador durante o processo de inicialização, indicando que a aplicação está a carregar e assegurando uma experiência de utilizador mais fluida e informativa.

```
<ProgressBar
    android:id="@+id/progressBar"
    style="?android:attr/progressBarStyle"
    android:layout_width="73dp"
    android:layout_height="61dp"
    android:layout_gravity="center"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView6" />
```

Figura 25 – Configuração do ProgressBar.

O código completo do Activity_inicio.xml encontra-se no Anexo I.

5.8.2.2. Activity_main.xml

O *layout activity_main.xml* é responsável por definir a tela principal da aplicação. Nesta, o utilizador pode estabelecer uma ligação com o dispositivo UWB, visualizar a distância entre a Âncora e a *Tag*, e aceder a outras funcionalidades, como a visualização de tabelas e a secção "Sobre".

A estrutura principal do *layout* é gerida por um *ConstraintLayout*, que facilita o posicionamento dos elementos visuais de forma relativa uns aos outros e às extremidades da tela. Esta abordagem permite criar uma interface adaptável a diferentes tamanhos de ecrã, garantindo uma apresentação consistente. O fundo da tela é personalizado com um recurso gráfico específico, definido através da propriedade *android:background="@drawable/new_layout"* , apresentado na figura 26.

```
xmlns:app="http://schemas.android.com/apk/res-auto"
xmlns:tools="http://schemas.android.com/tools"
android:layout_width="match_parent"
android:background="@drawable/new_layout"
android:layout_height="match_parent"
tools:context=".MainActivity">
```

Figura 26 – Configuração dos atributos do *layout*.

Um dos componentes essenciais desta tela é o botão de ligação, identificado por *@+id/connectButton* e apresentado na figura 27. Este botão permite ao utilizador iniciar a ligação com o dispositivo UWB e possui um estilo personalizado, com o fundo definido por *android:background="@drawable/connect_botao"*. Além disso, o botão exibe um ícone de *conexão Bluetooth*, posicionado à esquerda do texto, através da propriedade *android:drawableLeft="@drawable/baseline_bluetooth_connected_24"*, e contém o texto *"Connect"*, que é visível ao utilizador.

```
android:id="@+id/connectButton"
android:layout_width="225dp"
android:layout_height="71dp"
android:background="@drawable/connect_botao"
android:drawableLeft="@drawable/baseline_bluetooth_connected_24"
android:padding="6dp"
android:text="Connect"
android:textSize="30sp"
```

Figura 27 – Configuração do botão de ligação.

Outro componente importante é o *ImageView,* apresentado na figura 28, identificado por *@+id/imageView*, que exibe uma imagem associada à aplicação, reforçando a ligação visual e a temática da interface. Esta imagem é especificada pela propriedade *app:srcCompat="@drawable/bag_ips"*.

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="166dp"
    android:layout_height="107dp"
    android:layout_gravity="center"
    app:layout_constraintBottom_toTopOf="@+id/connectButton"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView2"
    app:srcCompat="@drawable/bag_ips" />
```

Figura 28 – Configuração da *ImageView.*

A medição da distância é apresentada em tempo-real através de um *TextView,* apresentado na figura 29, identificado por *@+id/rangeValue*. Inicialmente, este componente exibe o texto "---- m", definido pela propriedade *android:text="---- m"*, mas o valor é atualizado continuamente pela aplicação à medida que novas medições são realizadas.

```
<TextView
    android:id="@+id/rangeValue"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:text="---- m"
    android:textColor="@color/white"
    android:textSize="60dp"
```

Figura 29 – Configuração do *TextView* para indicação do valor da distância em tempo-real.

Para além disso, existe um botão que permite ao utilizador visualizar o histórico de medições numa tabela. Este botão, identificado por *@+id/viewTableButton,* apresentado na figura 30, exibe o texto *"View Table"* e utiliza um estilo de fundo personalizado através da propriedade *android:background="@drawable/back_texto".*

```
<Button
    android:id="@+id/viewTableButton"
    android:layout_width="262dp"
    android:layout_height="63dp"
    android:background="@drawable/back_texto"
    android:gravity="center"
    android:text="View Table"
    android:textSize="25dp"
```

Figura 30 – Configuração do botão para visualizar a tabela.

A tela principal também inclui um botão de *logout*, identificado por *@+id/buttonLogout* apresentado na figura 31, que está configurado para chamar o método *logout* ao ser clicado, permitindo ao utilizador terminar a sessão na aplicação. Este comportamento é definido pela propriedade *android:onClick="logaut".*

```
<Button
    android:id="@+id/buttonLogout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/connect_botao"
    android:onClick="logaut"
    android:text="Logout"
```

Figura 31 – Configuração do botão *logout.*

Outros elementos visuais da interface incluem um *TextView*, apresentado na figura 32, para exibir o título principal da aplicação, identificado por *@+id/textView2*, que contém o texto "LSUWB". Existe ainda um *TextView* para identificar a *Tag*, com o ID @+id/textView3, que exibe o texto "TAG1:". Outro *TextView*, identificado por *@+id/textView4*, descreve a medição apresentada na tela, com o texto "*Distance Between ANCHOR 1 and TAG*".

```
<TextView
    android:id="@+id/textView4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:text="Distance Between ANCHOR 1 and TAG"
    android:textColor="@color/black"
    android:textSize="31dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/connectButton"
```

Figura 32 – Configuração do *TextView* para exibir o título da distância entre a *Tag* e a Âncora.

Finalmente, o *layout* inclui um botão "Sobre" apresentado na figura 33, identificado por *@+id/button2*, que redireciona o utilizador para uma secção onde são apresentadas informações sobre a aplicação.

```
<Button
    android:id="@+id/button2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:background="@drawable/connect_botao"
    android:text="Sobre"
```

Figura 33 Configuração do botão sobre.

O código XML completo do layout *activity_main.xml* encontra-se no Anexo I.

5.8.2.3. Activity_principal.xml

O layout *activity_principal.xml* define a interface gráfica para a tela de registo de utilizadores na aplicação. Este *layout* é responsável por obter informações básicas do utilizador, como nome, e-mail e palavra-passe, e inclui um botão para submeter esses dados para validação. A estrutura do *layout* foi concebida para ser simples, intuitiva e funcional, utilizando um *LinearLayout* com orientação vertical, para organizar os elementos na tela de forma sequencial.

A raiz do *layout* é um *LinearLayout,* apresentado na figura 34, que organiza os elementos em uma coluna vertical. A orientação é definida como vertical, através da propriedade *android:orientation="vertical"*, e o *android:gravity="center"* garante que os elementos sejam centralizados horizontalmente na tela.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/new_layout"
    android:gravity="center"
    android:orientation="vertical"
    android:paddingLeft="32dp"
    android:paddingRight="32dp"
    tools:context=".PrincipalActivity">
```

Figura 34 – Configuração das propriedades do *layout*.

O *layout* inclui vários componentes essenciais. O primeiro deles é um *TextView*, com *o ID @+id/textView*, que exibe o título da aplicação na parte superior da tela. Este componente utiliza as

seguintes propriedades: *android:text="LSUWB"* apresentado na figura 35, para definir o texto exibido, *android:textColor="@color/white",* para definir a cor do texto como branca, e android:textSize="40sp", para ajustar o tamanho do texto, tornando-o proeminente.

```
<TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:fontFamily="@font/black_ops_one"
    android:gravity="center"
    android:text="LSUWB"
    android:textColor="@color/white"
    android:textSize="40sp" />
```

Figura 35 – Configuração do TextView com o nome da App.

Segue-se um *EditText* apresentado na figura 36, para a inserção do nome do utilizador, identificado por @+id/editTextNome. As suas propriedades incluem:

- *android:hint="Name"*, para mostrar o texto "*Name*" como orientação, indicando o tipo de dado esperado;
- *android:inputType="text"*, que especifica que o campo aceita entrada de texto normal;
- *android:drawableLeft="@drawable/baseline_perm_identity_24*", para adicionar um ícone à esquerda do campo de texto, relacionado à identidade do utilizador;
- *android:textColorHint="@color/white"*, para definir a cor do texto como branca.

```
    <EditText
        android:id="@+id/editTextNome"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:background="@drawable/back_texto"
        android:drawableLeft="@drawable/baseline_perm_identity_24"
        android:drawablePadding="6dp"
        android:ems="10"
        android:hint="Name"
        android:inputType="text"
        android:padding="10dp"
        android:textColor="@color/white"
        android:textColorHint="@color/white" />
```

Figura 36 – Configuração do *EditText.*

Para a inserção do e-mail, o *layout* inclui outro *EditText* com o ID *@+id/editTextEmail* apresentado na figura 37. Este componente exibe a Indicação "Email" (*android:hint="Email"*) e aceita tanto texto quanto endereços de e-mail (android:inputType="text|textWebEmailAddress").

Um ícone relacionado ao e-mail é adicionado à esquerda do campo (*android:drawableLeft ="@drawable/baseline_email_24"*).

```
<EditText
    android:id="@+id/editTextEmail"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:background="@drawable/back_texto"
    android:drawableLeft="@drawable/baseline_email_24"
    android:drawablePadding="6dp"
    android:ems="10"
    android:hint="Email"
```

Figura 37 – Configuração de um *EditText* do Email.

A inserção da palavra-passe é gerida por um *EditText* identificado por *@+id/editTextPass,* apresentado na figura 38. Este campo exibe a indicação "*Password*" (*android:hint="Password")* e é configurado para aceitar texto que será ocultado, sendo adequado para palavras-passe (*android:inputType= "text|textPassword*").

Para reforçar a segurança, um ícone de cadeado é adicionado à esquerda do campo (*android:drawableLeft="@drawable/baseline_lock_24"*).

```
<EditText
    android:id="@+id/editTextPass"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginBottom="8dp"
    android:background="@drawable/back_texto"
    android:drawableLeft="@drawable/baseline_lock_24"
    android:drawablePadding="6dp"
    android:ems="10"
```

Figura 38 – Configuração *EditText* do Password.

Por fim, o *layout* inclui um botão de registo, identificado por *@+id/buttomRegistar1* e apresentado na figura 39. O texto do botão é definido como "Registar" (*android:texto = "Registar"*), e o fundo personalizado é aplicado através de *android:background= "@drawable/back_botao"*. O botão está configurado para chamar o método *validarCampos* (*android:onClick= "validarCampos"*), o que permite ao utilizador submeter os dados preenchidos para validação e posterior registo na aplicação.

```
<Button
    android:id="@+id/buttomRegistar1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/back_botao"
    android:onClick="validarCampos"
    android:text="Registar"
```

Figura 39 – Configuração do botão registo.

O código XML completo do *layout activity_principal.xml* encontra-se no Anexo I.

5.8.2.4. Activity_secundario.xml

O *layout activity_secundario.xml* é utilizado para a interface de *login* da aplicação. Semelhante ao *layout activity_principal.xml*, este organiza os elementos numa estrutura simples e funcional, utilizando um *LinearLayout* com orientação vertical, para centralizar e alinhar os componentes de forma clara e

acessível ao utilizador.

Assim como no layout *activity_principal.xml*, o *LinearLayout* preenche toda a largura e altura da tela, com orientação vertical e centralização dos elementos. O fundo é personalizado com um recurso gráfico, e as margens internas são ajustadas para proporcionar um espaçamento adequado.

Os componentes principais deste *layout* também incluem *TextViews* e *EditTexts*, com funcionalidades e propriedades semelhantes às descritas anteriormente. Por exemplo, o *TextView* utilizado no topo da interface exibe o título da aplicação com as mesmas configurações de texto, cor e fonte, enquanto os campos de entrada de texto (*EditTexts*) para e-mail e palavra-passe são configurados para aceitar os mesmos tipos de dados e utilizar ícones representativos à esquerda dos campos.

A principal diferença neste *layout* é a função específica de cada botão. O botão de *login*, identificado por *@+id/buttomLoguin* (figura 40), permite que o utilizador submeta as suas credenciais de *login* para validação, utilizando o método *validarAutenticacao*. Já o botão de registo, identificado por *@+id/buttomRegirase* e apresentado na figura 40, direciona o utilizador para a tela de registo caso ainda não tenha uma conta, configurado para chamar o método registar, quando pressionado. Além disso, a cor do texto e o estilo de fundo deste botão são ajustados, para diferenciar a função de registo de *login*.

```
<Button
    android:id="@+id/buttomLoguin"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/back_botao"
    android:onClick="validarAutenticacao"
    android:text="Entrar"
    android:textColor="@color/white"
    android:textSize="20sp" />

<Button
    android:id="@+id/buttomRegirase"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="8dp"
    android:background="@drawable/bavkg_regis"
    android:onClick="registar"
    android:text="Registar"
    android:textColor="@color/yelow"
    android:textSize="20sp" />
```

Figura 40 – Configurações dos botões entrar e registar.

O código XML completo do *layout activity_secundario.xml* encontra-se no Anexo I.

5.8.2.5. Activity_table.xml

O *layout activity_table.xml* define uma interface simples, que consiste numa tabela inserida dentro de um *ScrollView* e de um botão "*Reset*". Esta interface é projetada para exibir dados tabulares, permitindo ao utilizador arrastar a tabela para visualizar os dados, e também oferece a funcionalidade de redefinir esses dados através do botão "*Reset*".

O *ScrollView*, identificado por *@+id/scrollView*, ocupa a largura e a altura determinadas pelas restrições configuradas. As suas propriedades incluem *android:layout_width="0dp*" e *android:layout_height="0dp"*, o que permite que o *ScrollView* se ajuste às restrições de posicionamento em relação ao botão "*Reset*". A parte superior do *ScrollView* está alinhada ao topo do *ConstraintLayout (app:layout_ constraintTop_toTopOf="parent"),* enquanto a parte inferior está alinhada à parte superior do botão "*Reset*" (*app:layout_constraintBottom_toTopOf= "@+id/resetButton*"). A propriedade *app:layout_constraintStart_toStartOf= "parent"* e *app: layout_constraintEnd_toEndOf="parent"* assegura que o *ScrollView* esteja centralizado horizontalmente.

Dentro do *ScrollView* apresentado na figura 41, a tabela é gerida por um *TableLayout*, identificado por *@+id/tableLayout*, que exibe os dados em formato tabular. A largura da tabela é definida para ocupar toda a largura disponível na tela (*android:layout_width ="match_parent"*), enquanto a altura é ajustada ao conteúdo da tabela através de *android:layout_height ="wrap_content"*. Para melhorar a estética e a legibilidade, é aplicado um espaçamento interno com a propriedade *android:padding ="16dp"*. A propriedade *android:stretchColumns= "1"* é utilizada para esticar a segunda coluna, distribuindo o conteúdo uniformemente.

O botão "*Reset*" (figura 41), identificado por *@+id/resetButton*, está posicionado logo abaixo do *ScrollView*. As suas propriedades *android:layout_width ="145dp"* e *android:layout_height ="47dp"* definem as dimensões do botão. O fundo do botão é personalizado através de *android:background ="@drawable/back_texto"*, enquanto o texto "*Reset*" é definido por *android:texto ="Reset"*.

```
<ScrollView
    android:id="@+id/scrollView"
    android:layout_width="0dp"
    android:layout_height="0dp"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintBottom_toTopOf="@+id/resetButton"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintEnd_toEndOf="parent">

    <TableLayout
        android:id="@+id/tableLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="16dp"
        android:stretchColumns="1" />
</ScrollView>

<Button
    android:id="@+id/resetButton"
    android:layout_width="145dp"
    android:layout_height="47dp"
    android:background="@drawable/back_texto"
    android:textSize="20sp"
    android:text="Reset"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@id/scrollView" />
```

Figura 41 – Configuração do *ScrollView, TableLayout* e do Botão *Reset*.

O código XML completo do *layout* activity_table.xml encontra-se no Anexo I.

5.8.2.6. Activity_sobre.xml

O *layout activity_sobre.xml* é utilizado para apresentar uma mensagem de boas-vindas e uma breve descrição da aplicação de medição de distância em tempo-real, que utiliza a tecnologia UWB. Este *layout* combina um texto explicativo com uma imagem, proporcionando uma introdução visual à aplicação.

O *layout* principal é gerido por um *ConstraintLayout*, que ocupa toda a largura e altura disponíveis na tela e aplica um fundo personalizado. Os componentes são centralizados e organizados verticalmente, utilizando as mesmas propriedades de alinhamento descritas em *layouts* anteriores.

O *ImageView* apresentado na figura 42, identificado por *@+id/imageView*, exibe uma imagem relacionada ao tema da aplicação (logotipo IPS). A imagem é centralizada horizontalmente e posicionada em relação a um *TextView* anterior. As propriedades para as dimensões e posicionamento seguem o padrão já descrito para outros *ImageViews* em *layouts* anteriores.

```
<ImageView
    android:id="@+id/imageView"
    android:layout_width="251dp"
    android:layout_height="131dp"
    android:layout_gravity="center"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="@+id/textView9"
    app:layout_constraintVertical_bias="0.923"
    app:srcCompat="@drawable/bag_ips" />
```

Figura 42 – Configuração do *ImageView*.

O *TextView*, identificado por *@+id/textView9* e identificado na figura 43, exibe uma descrição informativa sobre a aplicação. O texto é justificado e o espaçamento entre linhas é ajustado para melhorar a legibilidade. O estilo da fonte é configurado para negrito, com o branco e um tamanho que assegura destaque visual. O posicionamento do *TextView* no *layout* segue o mesmo padrão de centralização utilizado em outros *layouts*.

```
    <TextView
        android:id="@+id/textView9"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginLeft="10dp"
        android:layout_marginTop="10dp"
        android:layout_marginRight="10dp"
        android:justificationMode="inter_word"
        android:lineSpacingMultiplier="1.2"
        android:text="Bem-vindo à nossa aplicação de medição de distância
em tempo real, utilizando tecnologia UWB (Ultra-Wideband). Este sistema foi
desenvolvido no âmbito da disciplina Dissertação/Projeto em CSC, com o
objetivo de fornecer medições precisas de distância através da utilização
de ÂNCORAs e TAGs. Utilizamos módulos DMW1000 em conjunto com as placas
XIAO ESP32C3 para garantir medições de alta precisão. A nossa aplicação
permite ao utilizador obter informações de distância em tempo real. A
```

```
aplicação foi desenhada para ser fácil de utilizar, proporcionando uma
experiência de utilizador eficiente e agradável"
        android:textColor="@color/white"
        android:textSize="20sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.842"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />
```

Figura 43 – Configuração do *TextView* .com a descrição da App.

O código XML completo do *layout* activity_sobre.xml pode ser consultado no Anexo I.

5.8.3. Drawables

Os *drawables* são recursos gráficos que podem ser aplicados a elementos visuais no *layout* de uma App Android. Estes recursos podem ser imagens simples (como PNGs ou JPEGs) ou arquivos XML que definem formas, gradientes, limites e outras características visuais. Para melhorar o desenho dos *layouts* e tornar a aplicação visualmente mais apelativa e funcional, foi necessário definir e utilizar recursos de *drawable* personalizados. Estes *drawables* são essenciais para aplicar efeitos visuais, como fundos, bordas arredondadas, gradientes, e para personalizar a aparência de botões, campos de texto e outros elementos da interface. No desenvolvimento da aplicação, foram criados diferentes tipos de *drawables* para personalizar a aparência dos *layouts*.

5.8.3.1. Descrição dos Drawables Utilizados:

- ***"bag_ips.png"***
  Trata-se de uma imagem PNG, que contém o logotipo do Instituto Politécnico de Setúbal (IPS). Esta imagem foi utilizada para dar identidade visual ao projeto desenvolvido no âmbito do IPS. A imagem foi aplicada em um *ImageView*, no *layout* da atividade *SobreActivity*, reforçando o *branding* institucional.
- ***"back_texto.xml"***
  Este *drawable* é um arquivo XML, que define um fundo com cor sólida e limites arredondados. O branco *(@color/white*) proporciona um contraste suave com o texto inserido nos campos. Foi utilizado como fundo para *EditText* nos formulários de *login* e registo, garantindo uma aparência agradável, além de melhorar a legibilidade dos campos de texto.
- ***connect_botao.xml***
  É um *drawable* em XML, que cria um botão com cantos suavemente arredondados (80dp), preenchido com uma cor semi-transparente branca (#22FFFFFF). Foi aplicado como fundo para botões que requerem destaque suave, mantendo a atenção do utilizador, sem sobrecarregar a interface.

- ***back_botao.xml***

  Este *drawable* XML define um fundo transparente com cantos arredondados (10dp) e uma borda fina de 1dp na cor azul (#c9dbe9). Foi utilizado para criar botões discretos, mas ainda assim destacados, mantendo uma aparência moderna e minimalista. A borda clara ajuda a delinear o botão sem dominá-lo.
- ***bavkg_regis.xml***

  É um *drawable* simples, que define um fundo branco sólido com cantos arredondados (10dp). Este recurso foi utilizado para botões que requerem um destaque claro e imediato, como o botão de registo, proporcionando um contraste visual adequado.

5.8.4. Activities

No desenvolvimento de aplicações Android, as *Activities* são componentes essenciais, que representam uma única tela com a interface do utilizador. Cada *Activity* no seu aplicativo desempenha um papel específico e contribui para a experiência geral do utilizador. Seguidamente é feita uma descrição detalhada das principais *Activities* da aplicação, suas funções e como elas interagem.

5.8.4.1. InicioActivity

A *InicioActivity* serve como a tela de abertura da aplicação, melhorando a experiência do utilizador ao fornecer uma introdução visual ao ser iniciada a App. A utilização de um *Handler* com *postDelayed* é uma técnica comum para exibir um ecrã de abertura antes de transitar para a interface principal do utilizador, que neste caso é a *SecundarioActivity*. Esta abordagem não só melhora a experiência do utilizador, como também permite que a aplicação carregue recursos importantes em segundo plano, preparando o sistema para a próxima tela.

Declaração do Pacote:

- O pacote com.example.agodist apresentado na figura 44 define o pacote onde a classe está localizada, ajudando a organizar o código e evitando conflitos de nome.

```
package com.example.agodist;

```

Figura 44 – Declaração de pacote da App no Android.

Importações:

- *AppCompatActivity* apresentado na figura 45: Importa a classe base para todas as *Activities* que utilizam as características do *AppCompat*, garantindo a compatibilidade com versões mais antigas do Android.
- *Intent* é utilizada para iniciar novas *Activities*. É um componente essencial para a navegação entre telas.
- *Bundle* consiste num pacote utilizado para passar dados entre diferentes componentes do sistema operativo Android. Embora não seja diretamente utilizado neste trecho de código, é essencial para atividades que recebem dados.
- *Handler* permite o agendamento de mensagens ou a execução de códigos num *thread*, sendo

utilizado para retardar a transição entre telas.

```
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
```

Figura 45 – Importação de bibliotecas.

Método *onCreate*:

- *protected void onCreate(Bundle savedInstanceState)* apresentado na figura 46, é um método de *callback* do ciclo de vida da *Activity*, chamado quando a *Activity* é criada pela primeira vez.
- *super.onCreate(savedInstanceState)* chama a implementação da superclasse para completar a criação da *Activity*.
- *setContentView(R.layout.activity_inicio):* define o *layout* XML a ser utilizado como a interface do utilizador para esta *Activity*.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_inicio);
```

Figura 46 Método *onCreate.*

***Handler* e *Runnable* para Atraso:**

- *new Handler().postDelayed(new Runnable*() {...}, 3000) apresentado na figura 47, cria um novo *Handler* e utiliza *postDelayed* para enfileirar um *Runnable*, que será executado após um atraso de 3000 milissegundos (3 segundos). Este método é usado para adicionar um atraso antes de iniciar uma ação, proporcionando tempo para que a tela de abertura seja exibida ao utilizador.
- *new Runnable*(): define o bloco de código a ser executado após o atraso.
- *public void run*(): método do *Runnable,* onde o código a ser executado é definido.
- *Intent* i = *new Intent*(*InicioActivity.this, SecundarioActivity.class*): Cria uma nova *Intent*, que descreve a *Activity* a ser iniciada.
- *startActivity*(i): Inicia a *Activity* especificada na *Intent*.

```
new Handler().postDelayed(new Runnable() {
    @Override
    public void run() {
        Intent i = new Intent(InicioActivity.this,SecundarioActivity.class);
        startActivity(i);
    }
},3000);
```

Figura 47 – Código para mudar ecrã com atraso 3 segundos.

No anexo I encontra-se o código XML completo do *InicioActivity.java*.

5.8.4.2. SecundarioActivity

A *SecundarioActivity* é uma componente crítica da aplicação, assumindo a responsabilidade primordial pela autenticação dos utilizadores. Este processo é essencial não apenas para a segurança

do sistema, mas também para personalizar e otimizar a experiência do utilizador, assegurando que dados sensíveis e funcionalidades críticas sejam acessíveis apenas a utilizadores verificados. A integração com o Firebase Authentication proporciona uma solução robusta e escalável.

5.8.4.2.1. Declaração de permissões, importações e dependências

As classes importadas facilitam a interface do utilizador, a autenticação e a gestão de tarefas assíncronas, refletindo a integração entre a interface Android e o *backend* de autenticação, gerido pelo Firebase. Na figura 48 são apresentadas as declarações de permissões, importações e dependências

```
package com.example.agodist;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.example.agodist.Util.ConfiguraBd;
import com.example.agodist.data.model.Utilizador;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseAuthInvalidCredentialsException;
import com.google.firebase.auth.FirebaseAuthInvalidUserException;
import com.google.firebase.auth.FirebaseUser;

import org.checkerframework.common.subtyping.qual.Bottom;
```

Figura 48 - Declaração de permissões, importações e dependências.

5.8.4.2.2. Inicialização e configuração da interface de utilizador

No método *onCreate,* apresentado na figura 49, é estabelecido o *layout* da *Secundario Activity* e inicializado o sistema de autenticação Firebase, preparando a base para a interação do utilizador e as operações de *backend*.

```
public class SecundarioActivity extends AppCompatActivity {

    EditText cappoEmail, campoSenha;
    Button botaoLogin;
    private  FirebaseAuth auth;



    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_secundario);
        auth = ConfiguraBd.Firebaseautenticacao();
```

```
        inicializarComponentes();
    }
private void inicializarComponentes(){
    cappoEmail = findViewById(R.id.editTextEmailLogin);
    campoSenha = findViewById(R.id.editTextPassLogin);
    botaoLogin = findViewById(R.id.buttomLoguin);
}
```

Figura 49 – Inicialização dos componentes de autenticação na *SecundarioActivity*.

5.8.4.2.3. Gestão de autenticação e navegação

A validação e a autenticação são realizadas verificando se os campos estão preenchidos antes de tentar autenticar o utilizador. Se os campos estiverem preenchidos corretamente, a autenticação prossegue; caso contrário, o utilizador é informado sobre a necessidade de preenchimento.

Na figura 50 é apresentada a parte do código responsável para validação da autenticação.

```
public void validarAutenticacao(View view){
    String email= cappoEmail.getText().toString();
    String senha= campoSenha.getText().toString();

    if(!email.isEmpty()){
        if (!senha.isEmpty()){

            Utilizador utilizador = new Utilizador();
            utilizador.setEmail(email);
            utilizador.setPassword(senha);

            logar(utilizador);

        }else {
            Toast.makeText(this, "Preencha a senha", Toast.LENGTH_SHORT).show();
        }

    }else {
        Toast.makeText(this, "Preencha o email", Toast.LENGTH_SHORT).show();
    }
}

private void logar(Utilizador utilizador) {
    auth.signInWithEmailAndPassword(
            utilizador.getEmail(),utilizador.getPassword()
    ).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
        @Override
        public void onComplete(@NonNull Task<AuthResult> task) {
            if (task.isSuccessful()){
                abrirHome();
            }else {
                String excessao="";
                try {
                    throw  task.getException();
                }catch (FirebaseAuthInvalidUserException e){
                    excessao = "Utilizador não está registado";
                }catch (FirebaseAuthInvalidCredentialsException e){
                    excessao = "Email ou Passwor incorreto";
                }catch (Exception e){
                    excessao = "Erro Login utilizador. Tenta outra
vez"+e.getMessage();
                    e.printStackTrace();
                }
                Toast.makeText(SecundarioActivity.this, excessao,
Toast.LENGTH_SHORT).show();
            }
        }
    });


}
```

Figura 50 – Método de validação de autenticação e *login* com *Firebase.*

5.8.4.2.4. Navegação contínua e gestão de sessão

Se o utilizador já estiver autenticado ao iniciar a *SecundarioActivity*, o sistema redireciona-se automaticamente para a *MainActivity* , apresentado na figura 51, evitando redundância no processo de *login*.

```
private void abrirHome() {
    Intent i = new Intent(SecundarioActivity.this,MainActivity.class);
    startActivity(i);
}


public void registar(View v){
    Intent i = new Intent(this,PrincipalActivity.class);
    startActivity(i);
}
```

Figura 51 – Método para redirecionar automaticamente para a *MainActivity* após autenticação.

O código Java completo do *SecundarioActivity.java* é apresentado no Anexo I.

5.8.4.3. PrincipalActivity

A *PrincipalActivity* serve como o ponto de entrada para novos utilizadores no sistema, garantindo que todos os dados necessários sejam corretamente recolhidos e validados antes do registo. Este processo é crucial, não apenas para a funcionalidade e segurança do sistema, mas também para assegurar o correto registo dos utilizadores.

5.8.4.3.1. Estrutura geral e importações

As importações e dependências necessárias que estão apresentadas na figura 52, garantem a integração entre as interfaces de utilizador, o *Firebase* para autenticação, e a gestão de tarefas assíncronas. Estas bibliotecas são fundamentais para as operações de registo e gestão de erros.

```
package com.example.agodist;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.example.agodist.Util.ConfiguraBd;
import com.example.agodist.data.model.Utilizador;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseAuthInvalidCredentialsException;
import com.google.firebase.auth.FirebaseAuthUserCollisionException;
import com.google.firebase.auth.FirebaseAuthWeakPasswordException;
```

Figura 52 – Importação de pacotes e bibliotecas necessária na autenticação.

5.8.4.3.2. Inicialização e configuração da interface de utilizador

No método *onCreate,* o *layout* da *PrincipalActivity* é configurado e os elementos de interface de utilizador são associados às variáveis da classe, conforme apresentado na figura 53. O evento de clique para o botão de registo é configurado para chamar o método validarCampos quando pressionado.

```
public class PrincipalActivity extends AppCompatActivity {

    Utilizador utilizador;
    FirebaseAuth autenticacao;

    EditText campoNome, campoEmail, campoSenha;
    Button botaoRegistar; // Corrigido de Bottom para Button

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_principal);

        inicializar();

        botaoRegistar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                validarCampos(v);
            }
        });
    }

    private void inicializar() {
        campoNome = findViewById(R.id.editTextNome);
        campoEmail = findViewById(R.id.editTextEmail);
        campoSenha = findViewById(R.id.editTextPass);
        botaoRegistar = findViewById(R.id.buttomRegistar1); // Corrigido de
Bottom para Button
    }
```

Figura 53 – Inicialização dos componentes e validação de registo.

5.8.4.3.3. Gestão de registo de utilizadores

A validação e o registo verificam se os campos de nome, email e senha estão completos antes de tentar registar o utilizador no *Firebase* (figura 54). O código também inclui a gestão de erros, tratando diferentes tipos de falhas na autenticação e informando o utilizador sobre a natureza do problema, para que possa corrigi-lo.

```
private void validarCampos(View v) {
        String nome = campoNome.getText().toString();
        String email = campoEmail.getText().toString();
        String senha = campoSenha.getText().toString();

        if (!nome.isEmpty()) {
            if (!email.isEmpty()) {
                if (!senha.isEmpty()) {

                    utilizador = new Utilizador(); // Inicializado corretamente

                    utilizador.setNome(nome);
                    utilizador.setEmail(email);
                    utilizador.setPassword(senha);

                } else {
                    Toast.makeText(this, "Preencha a senha",
Toast.LENGTH_SHORT).show();
                }
            } else {
                Toast.makeText(this, "Preencha o email",
Toast.LENGTH_SHORT).show();
            }
        } else {
            Toast.makeText(this, "Preencha o nome", Toast.LENGTH_SHORT).show();
        }
    }

    private void registarUtilizador() {

        autenticacao = ConfiguraBd.Firebaseautenticacao();

        autenticacao.createUserWithEmailAndPassword(
                utilizador.getEmail(),utilizador.getPassword()
        ).addOnCompleteListener(this, new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()){
                    Toast.makeText(PrincipalActivity.this, "Registe utilizador com
sucesso", Toast.LENGTH_SHORT).show();
                }else {
                    String excecao = "";
                    try {
                      throw task.getException();
                    }catch (FirebaseAuthWeakPasswordException e){
                        excecao="Senha fraca ou padrão. Digita novamente";
                    }catch (FirebaseAuthInvalidCredentialsException e){
                        excecao="Digite um email valido";
                    }catch (FirebaseAuthUserCollisionException e){
                        excecao="Este email ja existe";
                    }catch (Exception e){
                        excecao="Erro Registo Utilizador"+ e.getMessage();
                        e.printStackTrace();
                    }
                    Toast.makeText(PrincipalActivity.this, excecao,
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

Figura 54 – Validação do registo do utilizador com o *Firebase*.

O código Java completo da *PrincipalActivity.java* pode ser consultado no Anexo I.

5.8.4.4. MainActivity

A MainActivity.java integra funcionalidades essenciais para o funcionamento da App, aplicando conceitos avançados em programação Android, gestão de serviços *Bluetooth*, e práticas de segurança e autenticação através do *Firebase*. A análise detalhada deste código justifica as escolhas tecnológicas do projeto e destaca a capacidade de implementar e gerir uma aplicação complexa e funcional. Esta abordagem detalhada fornece uma visão clara sobre como a *MainActivity* opera e interage com outras partes do sistema, reforçando com uma análise técnica profunda e aplicada.

5.8.4.4.1. Pacote e importações

O código começa com a definição do pacote *com.example.agodist*, que organiza as classes e evita conflitos de nomes. Em seguida, são importadas bibliotecas (figura 55) essenciais como as classes para comunicação com dispositivos BLE, como o ESP32C3, o *FirebaseAuth* para autenticação de utilizadores, e classes como *TextView*, *Button*, e *ActivityCompat*, para manipulação da interface do utilizador e gestão de permissões. Estas importações fornecem as funcionalidades necessárias para a aplicação, incluindo autenticação e comunicação via *Bluetooth*.

```
package com.example.agodist;

import android.Manifest;

import android.bluetooth.BluetoothAdapter;

import android.bluetooth.BluetoothGatt;

import android.bluetooth.BluetoothGattCallback;

import android.bluetooth.BluetoothGattCharacteristic;

import android.bluetooth.BluetoothGattDescriptor;

import android.bluetooth.BluetoothGattService;

import android.bluetooth.BluetoothManager;

import android.bluetooth.le.ScanCallback;

import android.bluetooth.le.ScanResult;

import android.content.Intent;

import android.content.pm.PackageManager;

import android.os.Bundle;

import android.os.Handler;

import android.os.Looper;

import android.util.Log;

import android.view.View;
```

Figura 55 – Importação de bibliotecas para o *Bluetooth* e *Firebase*.

5.8.4.4.2. Declaração da classe principal e variáveis

A classe principal *MainActivity* herda de *AppCompatActivity*, permitindo gerir o ciclo de vida da atividade e interagir com a interface do utilizador. Variáveis como *FirebaseAuth auth* são utilizadas para gerir a autenticação dos utilizadores, enquanto variáveis como *rangeValue*, *connectButton*, e *bluetoothAdapter* (figura 56) permitem a interação com a interface e a comunicação *Bluetooth*. Constantes como *SERVICE_UUID* e as *CHARACTERISTIC_UUID* identificam os serviços e características específicas para comunicação com o dispositivo BLE, sendo fundamentais para o funcionamento correto da aplicação.

```
public class MainActivity extends AppCompatActivity {

    private FirebaseAuth auth;

    private static final String TAG = "MainActivity";

    private TextView rangeValue;
    private Button connectButton;
    private Button viewTableButton;
    private Button buttomSobre;

    private BluetoothAdapter bluetoothAdapter;
    private BluetoothGatt bluetoothGatt;

    private static final String ESP32_DEVICE_NAME = "ESP32_Ranging";
    private static final UUID SERVICE_UUID = UUID.fromString("00001101-
0000-1000-8000-00805F9B34FB");
    private static final UUID CHARACTERISTIC_UUID =
UUID.fromString("00002101-0000-1000-8000-00805F9B34FB");
    private static final UUID CLIENT_CHARACTERISTIC_CONFIG =
UUID.fromString("00002902-0000-1000-8000-00805f9b34fb");

    private static final int PERMISSION_REQUEST_CODE = 1;

    private List<RangeRecord> rangeHistory = new ArrayList<>(); // Lista
para armazenar o histórico
    private Handler handler = new Handler(Looper.getMainLooper());
    private Runnable recordRunnable = new Runnable() {
        @Override
        public void run() {
            // Adiciona um novo registro com data e hora atual
            String currentDate = new SimpleDateFormat("yyyy-MM-dd",
Locale.getDefault()).format(new Date());
            String currentTime = new SimpleDateFormat("HH:mm:ss",
Locale.getDefault()).format(new Date());
            String currentRange = rangeValue.getText().toString();

            rangeHistory.add(new RangeRecord(currentDate, currentTime,
currentRange));

            // Envia um broadcast para atualizar a tabela
            Intent intent = new Intent("com.example.agodist.RANGE_UPDATE");
            intent.putExtra("date", currentDate);
            intent.putExtra("time", currentTime);
            intent.putExtra("range", currentRange);
            sendBroadcast(intent);

            handler.postDelayed(this, 10000); // Executa novamente após 10
segundos
        }
    };
```

Figura 56 – Declaração e inicialização das variáveis do *Bluetooth* e *Firebase*.

5.8.4.4.3. Método “*onCreate*”

O método *onCreate* apresentado na figura 57 é o ponto de entrada da atividade, responsável por configurar a interface do utilizador e inicializar os componentes necessários. O *layout* é definido através de *setContentView(R.layout.activity_main)*, enquanto a autenticação *Firebase* é inicializada com *auth*

= *ConfiguraBd.Firebaseautenticacao()*. O código verifica se o *Bluetooth* está disponível e ativado, configurando os botões para responder aos cliques dos utilizadores, como iniciar a deteção de dispositivos BLE ou navegar para outra atividade.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    //cofiguração botão logout

    auth = ConfiguraBd.Firebaseautenticacao();

    rangeValue = findViewById(R.id.rangeValue);
    connectButton = findViewById(R.id.connectButton);
    viewTableButton = findViewById(R.id.viewTableButton);
    buttomSobre = findViewById(R.id.button2);

    buttomSobre.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent1 = new Intent(MainActivity.this,
SobreActivity.class);
            startActivity(intent1);
        }
    });




    BluetoothManager bluetoothManager = (BluetoothManager)
getSystemService(BLUETOOTH_SERVICE);
    bluetoothAdapter = bluetoothManager.getAdapter();

    if (bluetoothAdapter == null || !bluetoothAdapter.isEnabled()) {
        Toast.makeText(this, "Bluetooth is not supported or disabled",
Toast.LENGTH_LONG).show();
        return;
    }


    viewTableButton.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View v) {
            Intent intent = new Intent(MainActivity.this,
TableActivity.class);
            intent.putExtra("rangeHistory", (ArrayList<RangeRecord>)
rangeHistory);
            startActivity(intent);
        }
    });

    if (!checkPermissions()) {
        requestPermissions();
    }
}
```

Figura 57 – Método *onCreate.*

5.8.4.4.4. Verificação e solicitação de permissões

Antes de executar ações que envolvem *Bluetooth* ou localização, o método *checkPermissions* verifica se a aplicação possui as permissões necessárias (figura 58), solicitando-as ao utilizador caso não tenham sido concedidas (figura 59). Este processo garante que a aplicação cumpra as políticas de privacidade do Android.

```
private boolean checkPermissions() {
    boolean permissionsGranted = ContextCompat.checkSelfPermission(this,
Manifest.permission.BLUETOOTH) == PackageManager.PERMISSION_GRANTED &&
            ContextCompat.checkSelfPermission(this,
Manifest.permission.BLUETOOTH_ADMIN) == PackageManager.PERMISSION_GRANTED
&&
            ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED;

    return permissionsGranted;
}
```

Figura 58 – Verificação permissões do *Bluetooth.*

```
private void requestPermissions() {
    ActivityCompat.requestPermissions(this, new String[]{
            Manifest.permission.BLUETOOTH,
            Manifest.permission.BLUETOOTH_ADMIN,
            Manifest.permission.ACCESS_FINE_LOCATION},
PERMISSION_REQUEST_CODE);
}
```

Figura 59 – Solicitação permissões do *Bluetooth.*

5.8.4.4.5. Ligação *Bluetooth* e *Scan Callback*

O método *startScanning* inicia a deteção de dispositivos BLE (figura 60), procurando especificamente pelo dispositivo ESP32C3. Quando detetado, o *callback scanCallback* para a deteção e inicia a ligação ao dispositivo através de *connectGatt*, processo central para a comunicação com o ESP32 e para a obtenção dos dados do sensor de distância.

```
private void startScanning() {
    if (checkPermissions()) {
        try {

bluetoothAdapter.getBluetoothLeScanner().startScan(scanCallback);
            connectButton.setText("Connecting...");
            Log.d(TAG, "Started scanning for devices");
        } catch (SecurityException e) {
            Toast.makeText(this, "Bluetooth scan permission denied",
Toast.LENGTH_SHORT).show();
        }
    } else {
        requestPermissions();
    }
}

private void stopScanning() {
    if (checkPermissions()) {
        try {

bluetoothAdapter.getBluetoothLeScanner().stopScan(scanCallback);
            Log.d(TAG, "Stopped scanning for devices");
        } catch (SecurityException e) {
            Toast.makeText(this, "Bluetooth scan permission denied",
Toast.LENGTH_SHORT).show();
        }
    } else {
        requestPermissions();
    }
}

private final ScanCallback scanCallback = new ScanCallback() {
    @Override
    public void onScanResult(int callbackType, ScanResult result) {
        if (result.getDevice() != null &&
ESP32_DEVICE_NAME.equals(result.getDevice().getName())) {
            stopScanning();
            bluetoothGatt =
result.getDevice().connectGatt(MainActivity.this, false, gattCallback);
            Log.d(TAG, "Connecting to GATT server");
        }
    }
};
```

Figura 60 – Iniciar a deteção de dispositivos BLE e conectar.

5.8.4.4.6. *Callback* GATT para gerenciamento de ligação

O *callback BluetoothGattCallback,* apresentado na figura 61, gere os eventos de ligação BLE, incluindo conexão, desconexão e descoberta de serviços no dispositivo. Após a ligação, o aplicativo ativa notificações para características relevantes, permitindo que o valor da leitura do sensor seja atualizado e exibido ao utilizador em tempo-real.

```
private final BluetoothGattCallback gattCallback = new BluetoothGattCallback() {
    @Override
    public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
        if (newState == BluetoothGatt.STATE_CONNECTED) {
            if (checkPermissions()) {
                try {
                    gatt.discoverServices();
                    Log.d(TAG, "Discovering GATT services");
                } catch (SecurityException e) {
                    Toast.makeText(MainActivity.this, "Bluetooth discover services permission
denied", Toast.LENGTH_SHORT).show();
                }
            } else {
                requestPermissions();
            }
        } else if (newState == BluetoothGatt.STATE_DISCONNECTED) {
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    disconnect();
                }
            });
            Log.d(TAG, "Disconnected from GATT server");
        }
    }

    @Override
    public void onServicesDiscovered(BluetoothGatt gatt, int status) {
        BluetoothGattService service = gatt.getService(SERVICE_UUID);
        if (service != null) {
            BluetoothGattCharacteristic characteristic =
service.getCharacteristic(CHARACTERISTIC_UUID);
            if (characteristic != null) {
                if (checkPermissions()) {
                    try {
true);

                    } catch (SecurityException e) {
                        Toast.makeText(MainActivity.this, "Bluetooth characteristic
notification permission denied", Toast.LENGTH_SHORT).show();
                    }
                } else {
                    requestPermissions();
                }
            }
        }
    }

    @Override
    public void onCharacteristicChanged(BluetoothGatt gatt, BluetoothGattCharacteristic
characteristic) {
        if (CHARACTERISTIC_UUID.equals(characteristic.getUuid())) {
            final String value = characteristic.getStringValue(0);
            runOnUiThread(new Runnable() {
                @Override
                public void run() {
                    rangeValue.setText(value + " m");
                    Log.d(TAG, "Characteristic changed: Range = " + value + " m");
                }
            });
        }
    }
};
```

Figura 61 – *Callback* GATT para gerenciamento de ligação.

5.8.4.4.7. Desconexão e limpeza de recursos

O método *disconnect* apresentado na figura 62 é chamado para encerrar a ligação *Bluetooth* de forma segura. Ele desliga o dispositivo através de *bluetoothGatt.disconnect*() e fecha a ligação com *bluetoothGatt.close*(). Este procedimento assegura a liberação correta dos recursos, prevendo perdas de memória e problemas correlacionados. Ao concluir a desconexão, o texto do botão de ligação é

redefinido para "*Connect*", e o campo de leitura é reiniciado para indicar que não há dados disponíveis.

```
private void disconnect() {
    if (checkPermissions()) {
        if (bluetoothGatt != null) {
            try {
                bluetoothGatt.disconnect();
                bluetoothGatt.close();
                bluetoothGatt = null;
                rangeValue.setText("---- m");
                connectButton.setText("Connect");
                handler.removeCallbacks(recordRunnable); // Remove o
callback ao desconectar
            } catch (SecurityException e) {
                Toast.makeText(this, "Bluetooth disconnect permission
denied", Toast.LENGTH_SHORT).show();
            }
        }
    } else {
        requestPermissions();
    }
}
```

Figura 62 – Método que garante a desconexão de um dispositivo.

5.8.4.4.8. *Callback* de permissões e finalização da atividade

O *callback onRequestPermissionsResult,* apresentado na figura 63, trata o resultado das solicitações de permissões, verificando se foram concedidas e iniciando a deteção *Bluetooth* caso necessário. Quando a atividade é finalizada, o método *onDestroy* fecha qualquer ligação *Bluetooth* ativa e remove os *callbacks* pendentes, garantindo que a aplicação não consome mais recursos após ser encerrada.

```
@Override
    public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
        if (requestCode == PERMISSION_REQUEST_CODE) {
            boolean allPermissionsGranted = true;
            for (int result : grantResults) {
                if (result != PackageManager.PERMISSION_GRANTED) {
                    allPermissionsGranted = false;
                    break;
                }
            }

            if (allPermissionsGranted) {
                startScanning();
            } else {
                Toast.makeText(this, "Permissions not granted",
Toast.LENGTH_SHORT).show();
            }
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (bluetoothGatt != null) {
            try {
                bluetoothGatt.close();
            } catch (SecurityException e) {
                Log.e(TAG, "Bluetooth close permission denied", e);
            }
            bluetoothGatt = null;
        }
        handler.removeCallbacks(recordRunnable); // Remove o callback ao
destruir a atividade
    }
}
```

Figura 63 – Método para gestão de permissões e encerramento de recurso do *Bluetooth*.

O código Java completo do *MainActivity.java* encontra-se no Anexo I.

5.8.4.5. TableActivity

A *TableActivity* é uma atividade dedicada a exibir o histórico de leituras de distância recolhidas pela aplicação. Estas leituras são apresentadas numa tabela organizada, onde cada linha corresponde a uma nova leitura, com informações sobre a data, hora e distância medida. Esta atividade permite que o utilizador visualize os dados armazenados e, se necessário, limpe o histórico de leituras. Além disso, a *TableActivity* é atualizada de 10 em 10 segundos, sempre que novas leituras são registadas, graças à utilização de um *BroadcastReceiver*, que recebe dados da atividade principal (*MainActivity*) e atualiza automaticamente a tabela. Abaixo, segue uma análise detalhada do código da *TableActivity.*

5.8.4.5.1. Pacote e importações

O pacote *com.example.agodist* mantém a estrutura de organização da aplicação, como explicado anteriormente. As classes importadas provêm da biblioteca Android (figura 64) e incluem componentes da interface de utilizador, como *TableLayout*, *TableRow*, *TextView*, *Button* e *ScrollView*, que são utilizados para construir e manipular a interface da tabela e dos botões. Além destes componentes, a classe *BroadcastReceiver* permite que a atividade receba notificações (*broadcasts*) enviadas pela *MainActivity*, atualizando a tabela em tempo-real sempre que novos dados são transmitidos.

```
package com.example.agodist;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.widget.Button;
import android.widget.ScrollView;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import java.util.ArrayList;
```

Figura 64 – Importação de pacotes e definição das classes principais.

5.8.4.5.2. Declaração da classe e variáveis

A classe *TableActivity* estende a *AppCompatActivity* apresentada na figura 65, permitindo utilizar o ciclo de vida da atividade e outros recursos do Android. Entre as principais variáveis, destaca-se a *tableLayout*, que diz respeito ao *layout* da tabela que exibirá os registos. A variável *rangeHistory* é uma lista de objetos *RangeRecord*, que contém o histórico das leituras de distância, data e hora, recebida da *MainActivity* através de um *Intent*. O *resetButton* é um botão que permite ao utilizador limpar todos os registos da tabela. Por fim, o *rangeUpdateReceiver* é um *BroadcastReceiver* que permite que a *TableActivity* seja atualizada em tempo-real, sempre que novos registos de distância são enviados pela *MainActivity*.

```
public class TableActivity extends AppCompatActivity {

    private TableLayout tableLayout;
    private ArrayList<RangeRecord> rangeHistory;
    private Button resetButton;

    private final BroadcastReceiver rangeUpdateReceiver = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            if
("com.example.agodist.RANGE_UPDATE".equals(intent.getAction())) {
                String date = intent.getStringExtra("date");
                String time = intent.getStringExtra("time");
                String range = intent.getStringExtra("range");
                RangeRecord record = new RangeRecord(date, time, range);
                rangeHistory.add(record);
                addTableRow(record);
            }
        }
    };
```

Figura 65 – Declaração de classes.

5.8.4.5.3. Receção de dados em tempo-real com *BroadcastReceiver*

A *TableActivity* é capaz de receber dados em tempo-real graças ao *rangeUpdateReceiver,* apresentado na figura 66. Quando um novo valor de distância é lido na *MainActivity*, essa informação é transmitida *(broadcast)* e recebida pela *TableActivity,* que adiciona uma nova linha à tabela com os dados atualizados. Para garantir que a atividade se encontra pronta para receber essas atualizações, o método *registerReceiver()* é chamado no *onCreate,* com um filtro que escuta *broadcasts* com a ação *"com.example.agodist.RANGE_UPDATE".*

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_table);

    tableLayout = findViewById(R.id.tableLayout);
    resetButton = findViewById(R.id.resetButton);

    rangeHistory = (ArrayList<RangeRecord>)
getIntent().getSerializableExtra("rangeHistory");

    updateTable();

    resetButton.setOnClickListener(v -> {
        rangeHistory.clear();
        tableLayout.removeAllViews();
        Toast.makeText(TableActivity.this, "Registros reiniciados",
Toast.LENGTH_SHORT).show();
    });

    registerReceiver(rangeUpdateReceiver, new
IntentFilter("com.example.agodist.RANGE_UPDATE"));
}
```

Figura 66 – Implementação do método responsável pela inicialização e pela gestão de eventos.

5.8.4.5.4. Método *onCreate*

No método *onCreate*, a configuração da interface da *TableActivity* é realizada utilizando o método *setContentView(R.layout.activity_table)*. A tabela é carregada através de *findViewById (R.id.tableLayout)*, e o botão de reset também é inicializado nesse processo. Quanto ao histórico de leituras, a lista de registos de distância (*rangeHistory*) é transferida da *MainActivity* para a *TableActivity* através de um *Intent*. O histórico de leituras é então obtido, utilizando o método *getIntent().getSerializableExtra("rangeHistory")*.

5.8.4.5.5. Adicionar linhas à tabela (*addTableRow*)

O método *addTableRow()* (apresentado na figura 67) é responsável por adicionar linhas à tabela, onde cada registo de leitura é representado como uma linha individual. Este método cria uma nova linha (*TableRow*) e, para cada registo, adiciona três *TextViews*, que mostram a data, a hora e a distância correspondente. Para melhorar a legibilidade, cada *TextView* dentro da linha da tabela é configurado com espaçamento (*setPadding*), e é aplicado um fundo com borda (*setBackgroundResource(R.drawable.cell_border)*), criando uma separação visual clara entre as células.

```
private void addTableRow(RangeRecord record) {
    TableRow tableRow = new TableRow(this);

    TextView dateView = new TextView(this);
    dateView.setText(record.getDate());
    dateView.setPadding(16, 16, 16, 16);
    dateView.setBackgroundResource(R.drawable.cell_border);

    TextView timeView = new TextView(this);
    timeView.setText(record.getTime());
    timeView.setPadding(16, 16, 16, 16);
    timeView.setBackgroundResource(R.drawable.cell_border);

    TextView rangeView = new TextView(this);
    rangeView.setText(record.getRange());
    rangeView.setPadding(16, 16, 16, 16);
    rangeView.setBackgroundResource(R.drawable.cell_border);

    tableRow.addView(dateView);
    tableRow.addView(timeView);
    tableRow.addView(rangeView);

    tableLayout.addView(tableRow);
}
```

Figura 67 – Método *addTableRow* para adicionar uma nova linha à tabela.

5.8.4.5.6. Botão *reset* (*resetButton*)

O botão de *reset* (*resetButton*) apresentado na figura 68 tem a funcionalidade de limpar o histórico de leituras e remover todas as linhas da tabela. Quando o botão é pressionado, o método *rangeHistory.clear()* é chamado para limpar a lista, enquanto o *tableLayout.removeAllViews()* remove todas as linhas visíveis da tabela.

```
resetButton.setOnClickListener(v -> {
    rangeHistory.clear();
    tableLayout.removeAllViews();
    Toast.makeText(TableActivity.this, "Registros reiniciados",
Toast.LENGTH_SHORT).show();
});
```

Figura 68 – Configuração do botão de reiniciar.

5.8.4.5.7. Criar e estilizar *TextViews* para a tabela (*createTextView*)

O método *createTextView(),* apresentado na figura 69, é uma função de utilidade que simplifica a criação de *TextViews* para a tabela. Este método recebe um texto como parâmetro, cria um *TextView* com o texto fornecido e aplica-lhe o espaçamento necessário (*setPadding*) para melhorar a legibilidade.

```
private TextView createTextView(String text) {
    TextView textView = new TextView(this);
    textView.setText(text);
    textView.setPadding(8, 8, 8, 8);
    return textView;
}
```

Figura 69 – Método responsável pela criação de elementos de texto de forma dinâmica.

5.8.4.5.8. Método *onDestroy*

A limpeza de recursos que está apresentada na figura 70 é uma das etapas mais importantes que ocorre quando a atividade é finalizada, como por exemplo quando o utilizador sai da *TableActivity*. O método *onDestroy* é chamado para realizar essa limpeza, e o *unregisterReceiver(rangeUpdateReceiver)* é utilizado para garantir que o *BroadcastReceiver* seja corretamente desativado, evitando que continue a receber atualizações desnecessárias após a atividade ter sido encerrada.

```
@Override
protected void onDestroy() {
    super.onDestroy();
    unregisterReceiver(rangeUpdateReceiver);
}
```

Figura 70 – Método para limpeza de recursos.

O código Java completo do *TableActivity.java* pode ser consultado no Anexo I.

5.8.4.6. SobreActivity

A *SobreActivity* é uma atividade simples, que exibe informações sobre a aplicação. Esta atividade é utilizada para mostrar detalhes como o nome da aplicação, versão, autores e outros dados relevantes para o utilizador.

5.8.4.6.1. Pacote e importações

O pacote *com.example.agodist* organiza a classe dentro da estrutura do projeto. A classe *SobreActivity* apresentada na figura 71 estende a *AppCompatActivity*, o que permite utilizar o ciclo de vida completo da atividade e acesar aos recursos do Android.

```
package com.example.agodist;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class SobreActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sobre);
    }
}
```

Figura 71 – Atividade responsável por mostrar a tela *SobreActivity* na App.

O código Java completo do arquivo SobreActivity.java está disponível no Anexo I. Neste trabalho, foram abordados detalhadamente os diversos aspetos do desenvolvimento da aplicação móvel, concebida para interagir com o sistema de localização UWB, permitindo ao utilizador visualizar, em tempo-real, a distância entre uma Tag e uma Âncora.

Este capítulo descreve uma visão abrangente do desenvolvimento da App, desde a conceção do projeto até à implementação das funcionalidades essenciais de comunicação e dados. A aplicação foi desenvolvida com foco na simplicidade de utilização, gestão eficaz de permissões e integração com o Firebase, resultando num projeto de *software* final robusto, eficiente e de fácil utilização. A estrutura modular da aplicação, o design intuitivo das interfaces e a gestão eficiente dos recursos e permissões garantiram uma experiência de utilizador rica e de fácil utilização.

Capítulo 6
Testes ao sistema

6.1. Introdução

Este capítulo descreve os testes realizados ao sistema de medição de distâncias baseado na tecnologia UWB. Os testes tiveram como objetivo validar a funcionalidade e o desempenho do sistema, avaliar a precisão das medições de distância, e verificar a estabilidade da comunicação BLE entre a *Tag* e a aplicação Android. Para garantir a fiabilidade do sistema, foram conduzidos diversos cenários de teste, incluindo medições em diferentes distâncias e em ambientes variados.

6.2. Testes de medição de distância

Os testes de medição de distância foram realizados para validar a precisão das leituras entre a *Tag* e a Âncora. Utilizando a tecnologia UWB, o sistema demonstrou ser capaz de medir distâncias com precisão até um máximo de aproximadamente 15 metros, que foi o limite físico imposto durante os testes. A precisão da medição, com um desvio mínimo de alguns centímetros, comprovou que o sistema é robusto e adequado para a localização e monitorização. As tabelas 6 e 7 mostram os resultados dos testes realizados.

Tabela 6 – Valores recomendados de energia TX, para frequência 16 MHz.

Canal TX	Valor recomendado para potência de transmissão, para a frequência 16 MHz (quando a potência inteligente está ativada)	Valor recomendado para potência de transmissão, para a frequência 16 MHz (quando a potência inteligente está desativada)	Distância (m)
1	0x15355575	0x75757575	8,67
2	0x15355575	0x75757575	9,10
3	0x0F2F4F6F	0x6F6F6F6F	7,67
4	0x1F1F3F5F	0x5F5F5F5F	11,80
5	0x0E082848	0x48484848	13,02
7	0x32527292	0x92929292	10,04

Tabela 7 - Valores recomendados de energia TX, para frequência 64 MHz.

Canal TX	Valor recomendado para potência de transmissão, para a frequência 64 MHz (quando a potência inteligente está ativada)	Valor recomendado para potência de transmissão, para a frequência 64 MHz (quando a potência inteligente está desativada)	Distância (m)
1	0x07274767	0x67676767	9,65
2	0x07274767	0x67676767	9,15
3	0x2B4B6B8B	0x8B8B8B8B	8,16
4	0x3A5A7A9A	0x9A9A9A9A	6,80
5	0x25456585	0x85858585	14,07
7	0x5171B1D1	0xD1D1D1D1	11,32

Após a execução dos testes, observou-se que com a funcionalidade de energia TX inteligente do DW1000 ativada {*void dwt_setsmarttxpower(int enable)*;} não existiram alterações significativas no valor de distância medido. Portanto, ficou configurado o canal 5 a 64 MHz para energia TX.

6.3. Estimativa da Relação Sinal/Ruído (SNR)

A Relação Sinal/Ruído (SNR) é uma medida essencial para avaliar a qualidade da comunicação entre a *Tag* e a Âncora. Nos testes realizados, a SNR foi estimada utilizando as leituras de potência do sinal recebido (*RX Power*) fornecidas pelo módulo DW1000. A Razão Sinal/Ruído é um termo para a razão entre as potências de um sinal contendo algum tipo de informação e o ruído de fundo (Wikipédia, 2024):

$$SNR = \frac{P_{\text{sinal}}}{P_{\text{ruido}}} \tag{6}$$

P_{sinal} é a potência do sinal recebido.

$P_{ruído}$ é a potência do ruído ambiente.

Como estas potências variam em várias ordens de magnitude, é comum representar a SNR em escala logarítmica, medindo em decibéis (dB). Sendo assim, a fórmula geral para a SNR em dB é obtida por:

$$SNR_{\text{dB}} = 10 \cdot \log_{10}\left(\frac{P_{\text{sinal}}}{P_{\text{ruido}}}\right) \tag{7}$$

Em relação ao modulo DW1000, as potências do sinal (P_{sinal}) e do ruido $(P_{\text{ruído}})$ são fornecidas em dBm. A unidade dBm representa a potência em relação a 1 *miliwatt* (1 mW) de forma logarítmica:

$$P_{\text{dBm}} = 10 \cdot \log_{10}\left(\frac{P}{1\ \text{mW}}\right) \tag{8}$$

Se a potência do sinal P_{sinal} e a potência do ruído $P_{ruído}$ são ambas representada em dBm, pode-se representar cada potência como:

$$P_{\text{sinal (dBm)}} = 10 \cdot \log_{10}\left(\frac{P_{\text{sinal}}}{1\ \text{mW}}\right) \tag{9}$$

$$P_{\text{ruído (dBm)}} = 10 \cdot \log_{10}\left(\frac{P_{\text{ruído}}}{1\ \text{mW}}\right) \tag{10}$$

Como de (6):

$$SNR = \frac{P_{\text{sinal}}}{P_{\text{ruido}}} = \frac{\frac{P_{\text{sinal}}}{1\ \text{mW}}}{\frac{P_{\text{ruido}}}{1\ \text{mW}}} \tag{11}$$

De (7), (9) e (10) resultará que:

$$SNR_{\text{dB}} = 10 \cdot \log_{10}\left(\frac{\frac{P_{\text{sinal}}}{1\ \text{mW}}}{\frac{P_{\text{ruido}}}{1\ \text{mW}}}\right) = 10 \cdot \log_{10}\left(\frac{P_{\text{sinal}}}{1\ \text{mW}}\right) - 10 \cdot \log_{10}\left(\frac{P_{\text{ruido}}}{1\ \text{mW}}\right) = P_{\text{sinal (dBm)}} - P_{\text{ruído (dBm)}} \tag{12}$$

Ou seja:

$$SNR_{\text{dB}} = P_{\text{sinal (dBm)}} - P_{\text{ruído (dBm)}} \tag{13}$$

A tabela 8 apresenta os valores de SNR obtidos, com base em testes realizados.

Tabela 8 – Estimativa da Relação Sinal/Ruído (SNR) em dB.

Ambiente de Teste	**Potência do Sinal (P_{sinal})**	**Potência do Ruído ($P_{ruído}$)**	**Relação sinal-ruído (dB)**
Ambiente sem interferência	-45 dBm	-60 dBm	15 dB
Interferência ambiental	-50 dBm	-65 dBm	15 dB
Ambiente de alta interferência	-55 dBm	-70 dBm	15 dB

6.4. Estabilidade da conexão *Bluetooth Low Energy* (BLE)

Outro fator importante avaliado nos testes foi a estabilidade da comunicação BLE entre a *Tag* e a aplicação Android. Durante os testes, a ligação BLE foi mantida estável em todos os momentos, sem interrupções ou desconexões imprevistas. A taxa de transmissão de dados foi suficiente para que as medições de distância fossem atualizadas em tempo-real na aplicação móvel, proporcionando uma experiência de utilizador fluida. A comunicação BLE permaneceu estável durante todo o teste, sem interrupções;

6.5. Testes em diferentes ambientes

Os testes também incluíram medições em diferentes ambientes, como espaços interiores e exteriores, com o objetivo de avaliar o desempenho do sistema em cenários variados. Em ambientes abertos, o sistema manteve uma alta precisão de medição. Em espaços fechados, onde existiam obstáculos como paredes e portas, o sistema conseguiu manter a estabilidade da comunicação e medições precisas até cerca de 8 a 12 metros, dependendo do número de obstáculos presentes.

Os testes realizados demonstraram que o sistema desenvolvido de medição de distâncias é funcional, estável e eficaz para as finalidades propostas. As medições de distância foram precisas até 12 metros, e a comunicação BLE entre a *Tag* e a aplicação Android manteve-se estável e confiável ao longo de todos os testes. Estes resultados indicam que o sistema está apto para ser utilizado em cenários reais, cumprindo os requisitos de medição e comunicação definidos no projeto.

Capítulo 7
Conclusões

Considero que este trabalho alcançou com sucesso os objetivos inicialmente propostos, ao desenvolver um sistema de medição de distâncias baseado em tecnologia UWB, integrado com uma aplicação móvel desenvolvida na ferramenta Android Studio para monitorização em tempo-real. Os testes realizados confirmaram que o sistema apresenta elevada precisão na medição de distâncias e é capaz de comunicar via BLE, mesmo em ambientes dinâmicos. A implementação de um sistema de medição com estas características representa um contributo significativo para o campo dos sistemas de localização e monitorização, podendo ser utilizado para aplicações em áreas associadas à Indústria 4.0 e 5.0, como o rastreamento em tempo-real, a logística, a navegação interna e as interfaces humano-máquina. A integração entre o *hardware* e o *software* mostrou-se eficaz, facilitando a comunicação entre a *Tag* e a Âncora e proporcionando uma interface de utilizador intuitiva e funcional. Com os resultados obtidos verificou-se que a tecnologia UWB adotada é uma solução com elevado potencial para sistemas de localização com alta precisão.

O sistema desenvolvido pode ser futuramente ampliado, com a inclusão de mais Âncoras, permitindo a medição de distâncias em diversos eixos, e a consequente implementação de funcionalidades de localização tridimensional (3D). Além disso, a otimização do *firmware* para reduzir o consumo energético da *Tag* e a utilização de algoritmos de filtragem de sinal podem contribuir para aumentar ainda mais a precisão das medições. Sugere-se ainda a exploração de novas aplicações no âmbito da Indústria/Sociedade 4.0 e 5.0, onde a tecnologia UWB pode ser utilizada com sucesso para criar soluções inovadoras e eficientes.

Conclui-se, assim, que o desenvolvimento deste projeto atingiu plenamente os seus objetivos, integrando com sucesso as várias funcionalidades e proporcionando um dispositivo final de elevada resolução e bom desempenho até cerca de 12 metros.

Bibliografia

1. Alarifi, A., Al-Salman, A., Alsaleh, M., et al. (2016). Ultra Wideband Indoor Positioning Technologies: Analysis and Recent Advances. *Sensors*, 16(5), 707.
2. Apple. (2021). AirTag Technology Overview. Apple Whitepaper.
3. Bartholomew, P. R., & Green, E. J. (2019). UWB vs BLE in IoT Applications. *IEEE IoT Magazine*, 6(2), 98-104.
4. Brown, T. (2018). Advanced Wireless Communication Modules. IEEE Communications Magazine, 56(7), 34-42.
5. Decarli, N., Rizzardi, A., Centenaro, M., & Zanella, A. (2021). Performance Evaluation of UWB Localization for Industrial IoT Applications. IEEE Internet of Things Journal, 8(3), 2029-2038.
6. Decawave. (2020). RTLS Implementation Guidelines. Decawave Technical Report.
7. Decawave. (2021). DW1000 User Manual. Decawave Documentation.
8. Fischer, T., & Marzullo, P. (2019). IoT Applications with BLE Mesh Networking. *Sensors*, 19(12), 2658.
9. Fontana, R. J., & Gunderson, S. J. (2002). Ultra-Wideband Precision Asset Location System. *IEEE Aerosp. Electron. Syst. Mag.*, 17(5), 14-17.
10. Gallagher, T., & Detweiler, C. (2013). Using BLE for Low-Power IoT Systems. *J. IoT Eng.*, 2(1), 45-58.
11. Geng, Y., Chen, J., & Pahlavan, K. (2014). Research on UWB Radio Localization for Industrial Applications. *IEEE Trans. Ind. Electron.*, 61(4), 2272-2280.
12. Gezici, S., Poor, H. V., & Giannakis, G. B. (2005). Localization via Ultra-Wideband Radios. *IEEE Signal Process. Mag.*, 22(4), 70-84.
13. Gollakota, S., Hassanieh, H., & Katabi, D. (2013). Ultra-Low-Power Backscatter Communication Using BLE. *ACM SIGCOMM*, 14(3), 181-192.
14. Gomez, C., Oller, J., & Paradells, J. (2012). Overview and Evaluation of BLE for IoT. *Sensors*, 12(8), 11734-11753.
15. IEEE Communications Society. (2018). Overview of RTLS Technologies. *IEEE Communications Magazine*, 56(6), 92-98.
16. Inpixon. (2023). The Role of UWB in Modern RTLS Systems. Inpixon Whitepaper.
17. Inpixon. (2024). Ultra-Wideband Technology. Inpixon.
18. Lemic, F., Handziski, V., & Wolisz, A. (2016). Experimental Evaluation of RF-based Indoor Localization Algorithms under RF Interference. *Wireless Commun. Mobile Comput.*, 16(8), 1102-1116.
19. MikroElektronika, UWB Click, disponível em: https://www.mikroe.com/uwb-click?srsltid=AfmBOorZ71Qpl8o1k2apxhBwT62wQKTQxqEiQ9MDbmObN3jywkTOpYNK

20. Molisch, A. F., Win, M. Z., & Zhang, J. (2006). Ultra-Wideband Systems: Opportunities and Challenges. *Proc. IEEE*, 94(7), 1169-1189.
21. Mullner, N., & Wieser, M. (2019). Comparative Analysis of UWB RTLS and GPS. *IEEE Aerosp. Electron. Syst.*, 25(3), 110-119.
22. Mullner, N., Retscher, G., & Moser, E. (2020). UWB positioning system performance in multipath environments. *Sensors*, 20(15), 4265.
23. Qorvo. (2020). DW1000 Datasheet: Ultra-Wideband (UWB) Transceiver Module.
24. Seeed Studio (2023), XIAO ESP32C3 Getting Started, Seeed Studio Wiki, disponível em: https://wiki.seeedstudio.com/XIAO_ESP32C3_Getting_Started/.
25. Seeed Studio. (2023). XIAO ESP32C3 Specification. Seeed Studio Documentation.
26. Smith, J., & Lee, K. (2021). UWB Systems for Precision Location and Data Communication: Design and Application. IEEE Journal of Communication Systems, 52(3), 200-215.
27. Yang, T., Zhang, M., & Luo, J. (2020). Secure BLE Communication in IoT Devices. *Journal of Network Security*, 18(5), 135-142.
28. Wikipédia. (2024). *Relação sinal-ruído*. Wikipédia
29. Zafari, F., Gkelias, A., & Leung, K. K. (2019). A Survey of Indoor Localization Systems and Technologies. IEEE Communications Surveys & Tutorials, 21(3), 2568-2599.
30. Zwirello, L., & Schipper, M. (2013). Trends in Indoor UWB Localization. *IEEE Antennas Propag. Mag.*, 55(4), 256-269.
31. Zwirello, L., Schipper, F., Harter, M., & Zwick, T. (2012). UWB Localization System for Indoor Applications: Concept, Realization and Analysis. Journal of Communications and Networks, 14(3), 286-295.

Anexo I

Código completo da aplicação Android

Este anexo apresenta o código completo desenvolvido para a aplicação Android, que foi criada através da ferramenta Android Studio. A aplicação permite a visualização em tempo-real das medições de distância realizadas entre a *Tag* e a Âncora, utilizando comunicação BLE para transmitir os dados do dispositivo de *hardware* para a interface móvel. O código inclui a implementação de todas as funcionalidades da aplicação, desde a autenticação de utilizadores via Firebase até à gestão da interface gráfica e à comunicação com os módulos UWB.

AndroidManifest.xml

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.agodist">

    <uses-permission android:name="android.permission.BLUETOOTH" />
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"
/>

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/Theme.AGODIST">
        <activity
            android:name=".SecundarioActivity"
            android:exported="false" />
        <activity
            android:name=".PrincipalActivity"
            android:exported="false" />
        <activity
            android:name=".SobreActivity"
            android:exported="false" />
        <activity
            android:name=".MainActivity"
            android:exported="false" />
        <activity
            android:name=".InicioActivity"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER"
/>
            </intent-filter>
        </activity>
        <activity android:name=".TableActivity" />

        <meta-data
            android:name="preloaded_fonts"
            android:resource="@array/preloaded_fonts" />
```

```
    </application>

</manifest>
```

Activity_inicio.xml

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    android:background="@drawable/new_layout"

    android:orientation="vertical"

    tools:context=".InicioActivity">



    <TextView

        android:id="@+id/textView5"

        android:layout_width="match_parent"

        android:layout_height="match_parent"

        android:gravity="center"

        android:text="Bem-Vindo"

        android:textSize="40dp"

        app:layout_constraintBottom_toBottomOf="parent"

        app:layout_constraintEnd_toEndOf="parent"

        app:layout_constraintHorizontal_bias="0.0"

        app:layout_constraintStart_toStartOf="parent"

        app:layout_constraintTop_toTopOf="parent"

        app:layout_constraintVertical_bias="0.0" />



    <ProgressBar

        android:id="@+id/progressBar"
```

```
        style="?android:attr/progressBarStyle"
        android:layout_width="73dp"
        android:layout_height="61dp"
        android:layout_gravity="center"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView6" />

    <ImageView
        android:id="@+id/imageView"
        android:layout_width="199dp"
        android:layout_height="116dp"
        android:layout_gravity="center"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.478"
        app:layout_constraintStart_toStartOf="@+id/textView5"
        app:layout_constraintTop_toTopOf="@+id/textView5"
        app:layout_constraintVertical_bias="0.026"
        app:srcCompat="@drawable/bag_ips" />

    <TextView
        android:id="@+id/textView6"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="@font/black_ops_one"
        android:text="LSUWB"
        android:textColor="@color/white"
        android:textSize="50sp"
```

```
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/imageView"
        app:layout_constraintVertical_bias="0.103" />


    <TextView
        android:id="@+id/textView7"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="MEEC-CSC"
        android:textStyle="bold"
        android:textColor="@color/white"
        android:textSize="20dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.504"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/progressBar"
        app:layout_constraintVertical_bias="0.362" />


    <TextView
        android:id="@+id/textView8"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="By: AGo"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.498"
```

```
        app:layout_constraintStart_toStartOf="parent"

        app:layout_constraintTop_toBottomOf="@+id/textView7"

        app:layout_constraintVertical_bias="0.202" />
```

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:background="@drawable/new_layout"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <Button
        android:id="@+id/connectButton"
        android:layout_width="225dp"
        android:layout_height="71dp"
        android:background="@drawable/connect_botao"
        android:drawableLeft="@drawable/baseline_bluetooth_connected_24"
        android:padding="6dp"
        android:text="Connect"
        android:textSize="30sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView2"
        app:layout_constraintVertical_bias="0.309"
        app:layout_constraintVertical_chainStyle="packed" />


    <ImageView
        android:id="@+id/imageView"
        android:layout_width="166dp"
        android:layout_height="107dp"
        android:layout_gravity="center"
        app:layout_constraintBottom_toTopOf="@+id/connectButton"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView2"
        app:srcCompat="@drawable/bag_ips" />

    <TextView
        android:id="@+id/rangeValue"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="---- m"
        android:textColor="@color/white"
        android:textSize="60dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.84"
        app:layout_constraintStart_toStartOf="parent"
```

```
        app:layout_constraintTop_toBottomOf="@+id/textView4"
        app:layout_constraintVertical_bias="0.171" />

    <Button
        android:id="@+id/viewTableButton"
        android:layout_width="262dp"
        android:layout_height="63dp"
        android:background="@drawable/back_texto"
        android:gravity="center"
        android:text="View Table"
        android:textSize="25dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.496"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/rangeValue"
        app:layout_constraintVertical_bias="0.47" />

    <Button
        android:id="@+id/buttonLogout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/connect_botao"
        android:onClick="logaut"
        android:text="Logout"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.059"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.023" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="@font/black_ops_one"
        android:gravity="center"
        android:text="LSUWB"
        android:textColor="@color/white"
        android:textSize="50sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.125" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:fontFamily="@font/dm_serif_display"
        android:gravity="center"
        android:text="TAG1:"
        android:textColor="@color/white"
        android:textSize="60dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toStartOf="@+id/rangeValue"
        app:layout_constraintHorizontal_bias="0.638"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView4"
```

```
        app:layout_constraintVertical_bias="0.174" />

    <TextView
        android:id="@+id/textView4"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:text="Distance Between ANCHOR 1 and TAG"
        android:textColor="@color/black"
        android:textSize="31dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/connectButton"
        app:layout_constraintVertical_bias="0.104" />

    <Button
        android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="@drawable/connect_botao"
        android:text="Sobre"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.923"
        app:layout_constraintStart_toEndOf="@+id/buttonLogout"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.023" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_principal.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/new_layout"
    android:gravity="center"
    android:orientation="vertical"
    android:paddingLeft="32dp"
    android:paddingRight="32dp"
    tools:context=".PrincipalActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:fontFamily="@font/black_ops_one"
        android:gravity="center"
        android:text="LSUWB"
        android:textColor="@color/white"
        android:textSize="40sp" />

    <EditText
        android:id="@+id/editTextNome"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:background="@drawable/back_texto"
        android:drawableLeft="@drawable/baseline_perm_identity_24"
        android:drawablePadding="6dp"
        android:ems="10"
        android:hint="Name"
        android:inputType="text"
        android:padding="10dp"
        android:textColor="@color/white"
        android:textColorHint="@color/white" />

    <EditText
        android:id="@+id/editTextEmail"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:background="@drawable/back_texto"
        android:drawableLeft="@drawable/baseline_email_24"
        android:drawablePadding="6dp"
        android:ems="10"
        android:hint="Email"
        android:inputType="text|textWebEmailAddress"
        android:padding="10dp"
        android:textColor="@color/white"
        android:textColorHint="@color/white" />

    <EditText
        android:id="@+id/editTextPass"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:background="@drawable/back_texto"
        android:drawableLeft="@drawable/baseline_lock_24"
        android:drawablePadding="6dp"
        android:ems="10"
        android:hint="Password"
        android:inputType="text|textPassword"
        android:padding="10dp"
        android:textColor="@color/white"
        android:textColorHint="@color/white" />

    <Button
        android:id="@+id/buttomRegistar1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/back_botao"
        android:onClick="validarCampos"
        android:text="Registar"
        android:textColor="@color/white"
        android:textSize="20sp" />
</LinearLayout>
```

activity_secundario.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
```

```
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/new_layout"
    android:gravity="center"
    android:orientation="vertical"
    android:paddingLeft="32dp"
    android:paddingRight="32dp"
    tools:context=".SecundarioActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="16dp"
        android:fontFamily="@font/black_ops_one"
        android:gravity="center"
        android:text="LSUWB"
        android:textColor="@color/white"
        android:textSize="40sp" />


    <EditText
        android:id="@+id/editTextEmailLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="12dp"
        android:layout_marginBottom="8dp"
        android:background="@drawable/back_texto"
        android:drawableLeft="@drawable/baseline_email_24"
        android:drawablePadding="6dp"
        android:ems="10"
        android:hint="Email"
        android:inputType="text|textWebEmailAddress"
        android:padding="10dp"
        android:textColor="@color/white"
        android:textColorHint="@color/white" />

    <EditText
        android:id="@+id/editTextPassLogin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:background="@drawable/back_texto"
        android:drawableLeft="@drawable/baseline_lock_24"
        android:drawablePadding="6dp"
        android:ems="10"
        android:hint="Password"
        android:inputType="text|textPassword"
        android:padding="10dp"
        android:textColor="@color/white"
        android:textColorHint="@color/white" />

    <Button
        android:id="@+id/buttomLoguin"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@drawable/back_botao"
        android:onClick="validarAutenticacao"
        android:text="Entrar"
        android:textColor="@color/white"
```

```
        android:textSize="20sp" />

    <Button
        android:id="@+id/buttomRegirase"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:background="@drawable/bavkg_regis"
        android:onClick="registar"
        android:text="Registar"
        android:textColor="@color/yelow"
        android:textSize="20sp" />

</LinearLayout>
```

layout activity_table.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/new_layout"
    tools:context=".TableActivity">

    <ScrollView
        android:id="@+id/scrollView"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toTopOf="@+id/resetButton"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent">

        <TableLayout
            android:id="@+id/tableLayout"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="16dp"
            android:stretchColumns="1" />
    </ScrollView>

    <Button
        android:id="@+id/resetButton"
        android:layout_width="145dp"
        android:layout_height="47dp"
        android:background="@drawable/back_texto"
        android:textSize="20sp"
        android:text="Reset"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@id/scrollView" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

layout activity_sobre.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="@drawable/new_layout"
    android:gravity="center"
    android:orientation="vertical"
    tools:context=".SobreActivity">


    <ImageView
        android:id="@+id/imageView"
        android:layout_width="251dp"
        android:layout_height="131dp"
        android:layout_gravity="center"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="@+id/textView9"
        app:layout_constraintVertical_bias="0.923"
        app:srcCompat="@drawable/bag_ips" />

    <TextView
        android:id="@+id/textView9"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginLeft="10dp"
        android:layout_marginTop="10dp"
        android:layout_marginRight="10dp"
        android:justificationMode="inter_word"
        android:lineSpacingMultiplier="1.2"
        android:text="Bem-vindo à nossa aplicação de medição de distância
em tempo real, utilizando tecnologia UWB (Ultra-Wideband). Este sistema foi
desenvolvido no âmbito da disciplina Dissertação/Projeto em CSC, com o
objetivo de fornecer medições precisas de distância através da utilização
de ÂNCORAs e TAGs. Utilizamos módulos DMW1000 em conjunto com as placas
XIAO ESP32C3 para garantir medições de alta precisão. A nossa aplicação
permite ao utilizador obter informações de distância em tempo real. A
aplicação foi desenhada para ser fácil de utilizar, proporcionando uma
experiência de utilizador eficiente e agradável"
        android:textColor="@color/white"
        android:textSize="20sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.842"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.0" />


</androidx.constraintlayout.widget.ConstraintLayout>
```

InicioActivity.java

```
package com.example.agodist;
```

```
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;

public class InicioActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_inicio);

        new Handler().postDelayed(new Runnable() {
            @Override
            public void run() {
                Intent i = new
Intent(InicioActivity.this,SecundarioActivity.class);
                startActivity(i);
            }
        },3000);
    }
}
```

SecundarioActivity.java

```
package com.example.agodist;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.example.agodist.Util.ConfiguraBd;
import com.example.agodist.data.model.Utilizador;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseAuthInvalidCredentialsException;
import com.google.firebase.auth.FirebaseAuthInvalidUserException;
import com.google.firebase.auth.FirebaseUser;

import org.checkerframework.common.subtyping.qual.Bottom;

public class SecundarioActivity extends AppCompatActivity {

    EditText cappoEmail, campoSenha;
    Button botaoLogin;
    private  FirebaseAuth auth;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_secundario);
        auth = ConfiguraBd.Firebaseautenticacao();

        inicializarComponentes();
    }

    public void validarAutenticacao(View view){
        String email= cappoEmail.getText().toString();
        String senha= campoSenha.getText().toString();

        if(!email.isEmpty()){
            if (!senha.isEmpty()){

                Utilizador utilizador = new Utilizador();
                utilizador.setEmail(email);
                utilizador.setPassword(senha);

                logar(utilizador);

            }else {
                Toast.makeText(this, "Preencha a senha",
Toast.LENGTH_SHORT).show();
            }

        }else {
            Toast.makeText(this, "Preencha o email",
Toast.LENGTH_SHORT).show();
        }
    }

    private void logar(Utilizador utilizador) {
        auth.signInWithEmailAndPassword(
                utilizador.getEmail(),utilizador.getPassword()
        ).addOnCompleteListener(new OnCompleteListener<AuthResult>() {
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()){
                    abrirHome();
                }else {
                    String excessao="";
                    try {
                        throw  task.getException();
                    }catch (FirebaseAuthInvalidUserException e){
                        excessao = "Utilizador não está registado";
                    }catch (FirebaseAuthInvalidCredentialsException e){
                        excessao = "Email ou Passwor incorreto";
                    }catch (Exception e){
                        excessao = "Erro Login utilizador. Tenta outra
vez"+e.getMessage();
                        e.printStackTrace();
                    }
                    Toast.makeText(SecundarioActivity.this, excessao,
Toast.LENGTH_SHORT).show();
                }
            }
        });
```

```
    }

    private void abrirHome() {
        Intent i = new Intent(SecundarioActivity.this,MainActivity.class);
        startActivity(i);
    }


    public void registar(View v){
        Intent i = new Intent(this,PrincipalActivity.class);
        startActivity(i);
    }


        @Override
            protected void  onStart() {
            super.onStart();
            FirebaseUser utilizadorAuth =auth.getCurrentUser();
            if (utilizadorAuth !=null){
                abrirHome();
            }
    }


    private void inicializarComponentes(){
        cappoEmail = findViewById(R.id.editTextEmailLogin);
        campoSenha = findViewById(R.id.editTextPassLogin);
        botaoLogin = findViewById(R.id.buttomLoguin);
    }



}
```

PrincipalActivity.java

```
package com.example.agodist;

import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import com.example.agodist.Util.ConfiguraBd;
import com.example.agodist.data.model.Utilizador;
import com.google.android.gms.tasks.OnCompleteListener;
import com.google.android.gms.tasks.Task;
import com.google.firebase.auth.AuthResult;
import com.google.firebase.auth.FirebaseAuth;
import com.google.firebase.auth.FirebaseAuthInvalidCredentialsException;
import com.google.firebase.auth.FirebaseAuthUserCollisionException;
import com.google.firebase.auth.FirebaseAuthWeakPasswordException;

public class PrincipalActivity extends AppCompatActivity {
```

```
    Utilizador utilizador;
    FirebaseAuth autenticacao;

    EditText campoNome, campoEmail, campoSenha;
    Button botaoRegistar; // Corrigido de Bottom para Button

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_principal);

        inicializar();

        botaoRegistar.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                validarCampos(v);
            }
        });
    }

    private void inicializar() {
        campoNome = findViewById(R.id.editTextNome);
        campoEmail = findViewById(R.id.editTextEmail);
        campoSenha = findViewById(R.id.editTextPass);
        botaoRegistar = findViewById(R.id.buttomRegistar1); // Corrigido de
Bottom para Button
    }

    private void validarCampos(View v) {
        String nome = campoNome.getText().toString();
        String email = campoEmail.getText().toString();
        String senha = campoSenha.getText().toString();

        if (!nome.isEmpty()) {
            if (!email.isEmpty()) {
                if (!senha.isEmpty()) {

                    utilizador = new Utilizador(); // Inicializado
corretamente

                    utilizador.setNome(nome);
                    utilizador.setEmail(email);
                    utilizador.setPassword(senha);

                    registarUtilizador();

                } else {
                    Toast.makeText(this, "Preencha a senha",
Toast.LENGTH_SHORT).show();
                }
            } else {
                Toast.makeText(this, "Preencha o email",
Toast.LENGTH_SHORT).show();
            }
        } else {
            Toast.makeText(this, "Preencha o nome",
Toast.LENGTH_SHORT).show();
        }
    }
```

```
    private void registarUtilizador() {

        autenticacao = ConfiguraBd.Firebaseautenticacao();

        autenticacao.createUserWithEmailAndPassword(
                utilizador.getEmail(),utilizador.getPassword()
        ).addOnCompleteListener(this, new OnCompleteListener<AuthResult>()
{
            @Override
            public void onComplete(@NonNull Task<AuthResult> task) {
                if (task.isSuccessful()){
                    Toast.makeText(PrincipalActivity.this, "Registe
utilizador com sucesso", Toast.LENGTH_SHORT).show();
                }else {
                    String excecao = "";
                    try {
                      throw task.getException();
                    }catch (FirebaseAuthWeakPasswordException e){
                        excecao="Senha fraca ou padrão. Digita novamente";
                    }catch (FirebaseAuthInvalidCredentialsException e){
                        excecao="Digite um email valido";
                    }catch (FirebaseAuthUserCollisionException e){
                        excecao="Este email ja existe";
                    }catch (Exception e){
                        excecao="Erro Registo Utilizador"+ e.getMessage();
                        e.printStackTrace();
                    }
                    Toast.makeText(PrincipalActivity.this, excecao,
Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

MainActivity.java

```
package com.example.agodist;

import android.Manifest;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothGatt;
import android.bluetooth.BluetoothGattCallback;
import android.bluetooth.BluetoothGattCharacteristic;
import android.bluetooth.BluetoothGattDescriptor;
import android.bluetooth.BluetoothGattService;
import android.bluetooth.BluetoothManager;
import android.bluetooth.le.ScanCallback;
import android.bluetooth.le.ScanResult;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
```

```
import android.widget.Toast;
import androidx.annotation.NonNull;
import androidx.appcompat.app.AppCompatActivity;
import androidx.core.app.ActivityCompat;
import androidx.core.content.ContextCompat;

import com.example.agodist.Util.ConfiguraBd;
import com.google.firebase.auth.FirebaseAuth;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
import java.util.List;
import java.util.Locale;
import java.util.UUID;




public class MainActivity extends AppCompatActivity {

    private FirebaseAuth auth;

    private static final String TAG = "MainActivity";

    private TextView rangeValue;
    private Button connectButton;
    private Button viewTableButton;
    private Button buttomSobre;

    private BluetoothAdapter bluetoothAdapter;
    private BluetoothGatt bluetoothGatt;

    private static final String ESP32_DEVICE_NAME = "ESP32_Ranging";
    private static final UUID SERVICE_UUID = UUID.fromString("00001101-
0000-1000-8000-00805F9B34FB");
    private static final UUID CHARACTERISTIC_UUID =
UUID.fromString("00002101-0000-1000-8000-00805F9B34FB");
    private static final UUID CLIENT_CHARACTERISTIC_CONFIG =
UUID.fromString("00002902-0000-1000-8000-00805f9b34fb");

    private static final int PERMISSION_REQUEST_CODE = 1;

    private List<RangeRecord> rangeHistory = new ArrayList<>(); // Lista
para armazenar o histórico
    private Handler handler = new Handler(Looper.getMainLooper());
    private Runnable recordRunnable = new Runnable() {
        @Override
        public void run() {
            // Adiciona um novo registro com data e hora atual
            String currentDate = new SimpleDateFormat("yyyy-MM-dd",
Locale.getDefault()).format(new Date());
            String currentTime = new SimpleDateFormat("HH:mm:ss",
Locale.getDefault()).format(new Date());
            String currentRange = rangeValue.getText().toString();

            rangeHistory.add(new RangeRecord(currentDate, currentTime,
currentRange));

            // Envia um broadcast para atualizar a tabela
```

```
            Intent intent = new Intent("com.example.agodist.RANGE_UPDATE");
            intent.putExtra("date", currentDate);
            intent.putExtra("time", currentTime);
            intent.putExtra("range", currentRange);
            sendBroadcast(intent);

            handler.postDelayed(this, 10000); // Executa novamente após 10
segundos
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        //cofiguração botão logout

        auth = ConfiguraBd.Firebaseautenticacao();

        rangeValue = findViewById(R.id.rangeValue);
        connectButton = findViewById(R.id.connectButton);
        viewTableButton = findViewById(R.id.viewTableButton);
        buttomSobre = findViewById(R.id.button2);

        buttomSobre.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent1 = new Intent(MainActivity.this,
SobreActivity.class);
                startActivity(intent1);
            }
        });




        BluetoothManager bluetoothManager = (BluetoothManager)
getSystemService(BLUETOOTH_SERVICE);
        bluetoothAdapter = bluetoothManager.getAdapter();

        if (bluetoothAdapter == null || !bluetoothAdapter.isEnabled()) {
            Toast.makeText(this, "Bluetooth is not supported or disabled",
Toast.LENGTH_LONG).show();
            return;
        }

        connectButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (bluetoothGatt == null) {
                    if (checkPermissions()) {
                        startScanning();
                    } else {
                        requestPermissions();
                    }
                } else {
                    disconnect();
                }
            }
        });
```

```
        viewTableButton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent intent = new Intent(MainActivity.this,
TableActivity.class);
                intent.putExtra("rangeHistory", (ArrayList<RangeRecord>)
rangeHistory);
                startActivity(intent);
            }
        });

        if (!checkPermissions()) {
            requestPermissions();
        }
    }


    //cofiguração botão logout
    public  void logaut(View view){
        try {
            auth.signOut();
            finish();
        }catch (Exception e){
            e.printStackTrace();
        }
    }




    private boolean checkPermissions() {
        boolean permissionsGranted =
ContextCompat.checkSelfPermission(this, Manifest.permission.BLUETOOTH) ==
PackageManager.PERMISSION_GRANTED &&
                ContextCompat.checkSelfPermission(this,
Manifest.permission.BLUETOOTH_ADMIN) == PackageManager.PERMISSION_GRANTED
&&
                ContextCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED;

        return permissionsGranted;
    }

    private void requestPermissions() {
        ActivityCompat.requestPermissions(this, new String[]{
                Manifest.permission.BLUETOOTH,
                Manifest.permission.BLUETOOTH_ADMIN,
                Manifest.permission.ACCESS_FINE_LOCATION},
PERMISSION_REQUEST_CODE);
    }

    private void startScanning() {
        if (checkPermissions()) {
            try {

bluetoothAdapter.getBluetoothLeScanner().startScan(scanCallback);
                connectButton.setText("Connecting...");
                Log.d(TAG, "Started scanning for devices");
            } catch (SecurityException e) {
```

```
                Toast.makeText(this, "Bluetooth scan permission denied",
Toast.LENGTH_SHORT).show();
            }
        } else {
            requestPermissions();
        }
    }

    private void stopScanning() {
        if (checkPermissions()) {
            try {

bluetoothAdapter.getBluetoothLeScanner().stopScan(scanCallback);
                Log.d(TAG, "Stopped scanning for devices");
            } catch (SecurityException e) {
                Toast.makeText(this, "Bluetooth scan permission denied",
Toast.LENGTH_SHORT).show();
            }
        } else {
            requestPermissions();
        }
    }

    private void disconnect() {
        if (checkPermissions()) {
            if (bluetoothGatt != null) {
                try {
                    bluetoothGatt.disconnect();
                    bluetoothGatt.close();
                    bluetoothGatt = null;
                    rangeValue.setText("---- m");
                    connectButton.setText("Connect");
                    handler.removeCallbacks(recordRunnable); // Remove o
callback ao desconectar
                } catch (SecurityException e) {
                    Toast.makeText(this, "Bluetooth disconnect permission
denied", Toast.LENGTH_SHORT).show();
                }
            }
        } else {
            requestPermissions();
        }
    }

    private final ScanCallback scanCallback = new ScanCallback() {
        @Override
        public void onScanResult(int callbackType, ScanResult result) {
            if (result.getDevice() != null &&
ESP32_DEVICE_NAME.equals(result.getDevice().getName())) {
                stopScanning();
                bluetoothGatt =
result.getDevice().connectGatt(MainActivity.this, false, gattCallback);
                Log.d(TAG, "Connecting to GATT server");
            }
        }
    };

    private final BluetoothGattCallback gattCallback = new
BluetoothGattCallback() {
        @Override
```

```
        public void onConnectionStateChange(BluetoothGatt gatt, int status,
int newState) {
            if (newState == BluetoothGatt.STATE_CONNECTED) {
                if (checkPermissions()) {
                    try {
                        gatt.discoverServices();
                        Log.d(TAG, "Discovering GATT services");
                    } catch (SecurityException e) {
                        Toast.makeText(MainActivity.this, "Bluetooth
discover services permission denied", Toast.LENGTH_SHORT).show();
                    }
                } else {
                    requestPermissions();
                }
            } else if (newState == BluetoothGatt.STATE_DISCONNECTED) {
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        disconnect();
                    }
                });
                Log.d(TAG, "Disconnected from GATT server");
            }
        }

        @Override
        public void onServicesDiscovered(BluetoothGatt gatt, int status) {
            BluetoothGattService service = gatt.getService(SERVICE_UUID);
            if (service != null) {
                BluetoothGattCharacteristic characteristic =
service.getCharacteristic(CHARACTERISTIC_UUID);
                if (characteristic != null) {
                    if (checkPermissions()) {
                        try {

gatt.setCharacteristicNotification(characteristic, true);
                            BluetoothGattDescriptor descriptor =
characteristic.getDescriptor(CLIENT_CHARACTERISTIC_CONFIG);
                            if (descriptor != null) {

descriptor.setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
                                gatt.writeDescriptor(descriptor);
                            }
                            connectButton.setText("Disconnect");
                            Log.d(TAG, "Characteristic notification set");
                            handler.post(recordRunnable); // Inicia o
callback ao conectar
                        } catch (SecurityException e) {
                            Toast.makeText(MainActivity.this, "Bluetooth
characteristic notification permission denied", Toast.LENGTH_SHORT).show();
                        }
                    } else {
                        requestPermissions();
                    }
                }
            }
        }

        @Override
        public void onCharacteristicChanged(BluetoothGatt gatt,
BluetoothGattCharacteristic characteristic) {
```

```
            if (CHARACTERISTIC_UUID.equals(characteristic.getUuid())) {
                final String value = characteristic.getStringValue(0);
                runOnUiThread(new Runnable() {
                    @Override
                    public void run() {
                        rangeValue.setText(value + " m");
                        Log.d(TAG, "Characteristic changed: Range = " +
value + " m");
                    }
                });
            }
        }
    };

    @Override
    public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
        super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
        if (requestCode == PERMISSION_REQUEST_CODE) {
            boolean allPermissionsGranted = true;
            for (int result : grantResults) {
                if (result != PackageManager.PERMISSION_GRANTED) {
                    allPermissionsGranted = false;
                    break;
                }
            }

            if (allPermissionsGranted) {
                startScanning();
            } else {
                Toast.makeText(this, "Permissions not granted",
Toast.LENGTH_SHORT).show();
            }
        }
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        if (bluetoothGatt != null) {
            try {
                bluetoothGatt.close();
            } catch (SecurityException e) {
                Log.e(TAG, "Bluetooth close permission denied", e);
            }
            bluetoothGatt = null;
        }
        handler.removeCallbacks(recordRunnable); // Remove o callback ao
destruir a atividade
    }
}
```

TableActivity.java

```
package com.example.agodist;

import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Bundle;
import android.os.Handler;
import android.os.Looper;
```

```
import android.widget.Button;
import android.widget.ScrollView;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

import java.util.ArrayList;

public class TableActivity extends AppCompatActivity {

    private TableLayout tableLayout;
    private ArrayList<RangeRecord> rangeHistory;
    private Button resetButton;

    private final BroadcastReceiver rangeUpdateReceiver = new
BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            if
("com.example.agodist.RANGE_UPDATE".equals(intent.getAction())) {
                String date = intent.getStringExtra("date");
                String time = intent.getStringExtra("time");
                String range = intent.getStringExtra("range");
                RangeRecord record = new RangeRecord(date, time, range);
                rangeHistory.add(record);
                addTableRow(record);
            }
        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_table);

        tableLayout = findViewById(R.id.tableLayout);
        resetButton = findViewById(R.id.resetButton);

        rangeHistory = (ArrayList<RangeRecord>)
getIntent().getSerializableExtra("rangeHistory");

        updateTable();

        resetButton.setOnClickListener(v -> {
            rangeHistory.clear();
            tableLayout.removeAllViews();
            Toast.makeText(TableActivity.this, "Registros reiniciados",
Toast.LENGTH_SHORT).show();
        });

        registerReceiver(rangeUpdateReceiver, new
IntentFilter("com.example.agodist.RANGE_UPDATE"));
    }

    private void updateTable() {
        tableLayout.removeAllViews(); // Remove todas as linhas existentes
        TableRow headerRow = new TableRow(this);
        headerRow.addView(createTextView("Data"));
```

```
        headerRow.addView(createTextView("Hora"));
        headerRow.addView(createTextView("Distância (m)"));
        tableLayout.addView(headerRow);

        for (RangeRecord record : rangeHistory) {
            addTableRow(record);
        }
    }

    private void addTableRow(RangeRecord record) {
        TableRow tableRow = new TableRow(this);

        TextView dateView = new TextView(this);
        dateView.setText(record.getDate());
        dateView.setPadding(16, 16, 16, 16);
        dateView.setBackgroundResource(R.drawable.cell_border);

        TextView timeView = new TextView(this);
        timeView.setText(record.getTime());
        timeView.setPadding(16, 16, 16, 16);
        timeView.setBackgroundResource(R.drawable.cell_border);

        TextView rangeView = new TextView(this);
        rangeView.setText(record.getRange());
        rangeView.setPadding(16, 16, 16, 16);
        rangeView.setBackgroundResource(R.drawable.cell_border);

        tableRow.addView(dateView);
        tableRow.addView(timeView);
        tableRow.addView(rangeView);

        tableLayout.addView(tableRow);
    }

    private TextView createTextView(String text) {
        TextView textView = new TextView(this);
        textView.setText(text);
        textView.setPadding(8, 8, 8, 8);
        return textView;
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        unregisterReceiver(rangeUpdateReceiver);
    }
}
```

SobreActivity.java

```
package com.example.agodist;

import androidx.appcompat.app.AppCompatActivity;

import android.os.Bundle;

public class SobreActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_sobre);
```

```
        }
}
```