

**Escola Universitària d'Enginyeria Tècnica
de Telecomunicació La Salle**

Treball Final de Grau

Grau en Enginyeria Informàtica

Bpoll: Local offline educational polling application

Angel Farre Echaburua

Daniel Amo Filvà

ACTA DE L'EXAMEN DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Angel Farre Echaburua

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

Offline transmission of data in a local and proximity-based environment, oriented for didactic use.

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL

VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

Abstract

In an educational environment, the staff in charge of lecturing students often makes use of either an external service or a third-party service. Usually, these services provide the desired feature but overlook some key aspects.

The two main problems with this approach are a lack of security and privacy. Regarding security, the aforementioned services usually establish a connection with their own remote servers, meaning the exchange of messages with a potentially unprotected server. This means the messages may carry potentially malicious payloads the final user might not be aware of. Regarding privacy, there are no guarantees that external services do not record the user's data, that the server has not been compromised with a man in the middle or that the current network is not under sniffing attacks.

Overall, using external services means the user is prone to suffer cyberattacks. Therefore, in this work, an application will be presented which is designed to send and receive data in a secure, offline, and proximity-based manner, thus increasing security measures.

The data chosen to transfer for this proof of concept are forms, a simple tool widely used in the educational field by both teacher and students, and the first steppingstone for the development of similar tools such as tracking attendance or examinations.

This application has been implemented with Google's Nearby connections API and written in dart via the flutter framework, with the aim to offer a cross-platform solution. Other technologies have been studied too, which will also be explained in detail, defining what features and shortcomings they offer.

The resulting flutter application allows for android users to make full use of the features defined, although iOS users are, as of now, not able to make use of Nearby's API, thus limiting its cross-platform capabilities.

Keywords: application, Google Nearby connections, cross-platform, local, features, technologies, flutter, security.

Resum

En l'entorn educacional, el personal al càrrec de la docència dels estudiants fa us de forma sovint de eines externes o de tercers. Normalment, aquests serveis proveeixen les funcionalitats desitjades, però obvien alguns elements claus.

Els dos problemes principals amb aquesta practica es la manca de seguretat i privacitat. En quan a la seguretat, els serveis prèviament mencionats normalment estableixen connexions amb els seus propis servidors, la qual cosa significa un intercanvi de missatges amb un servidor potencialment insegur. Això vol dir que els missatges poden portar adjunts components maliciosos de les quals el usuari no se'n assabenta. En quan a la privacitat, no hi ha cap mena de garantia de que els serveis externs no emmagatzemin les dades del usuari, que el servidor s'ha vist compromès amb un atac d'intermediari o que la xarxa emprada no es trobi sota l'atac d'un *sniffer*.

En general, fer us de serveis externs significa que l'usuari es susceptible a sofrir ciberatacs. Degut a aquest fet, en aquest treball es presentarà un aplicació que permetrà a l'usuari enviar i rebre dades de forma segura, local i basat en proximitat, incrementant d'aquesta forma la seguretat.

Les dades escollides a enviar per aquesta prova de concepte son formularis, una eina senzilla però utilitzada de forma molt amplia en tot l'àmbit educatiu per tant estudiants com professors, i la primera pedra sobre la que es poden edificar altres funcionalitats com seria el seguiment de assistència o examinacions.

Aquesta aplicació s'ha implementat amb l'API de Google Nearby connections i esta escrita en dart a traves del framework de flutter, amb l'objectiu de assolir una solució multi plataforma. També s'han estudiat altres tecnologies, que seran explicades en detall, definint quines funcionalitats i mancances presenten cada una.

L'aplicació de flutter resultant permet als usuaris d'android fer us complet de les funcionalitats definides, tot i que a dia d'avui, els usuaris de iOS no poden fer servir la API de Nearby connections, limitant d'aquesta forma la seva capacitat multi plataforma.

Paraules clau: aplicació, Google Nearby connections, multi plataforma, local, funcionalitats, tecnologies, flutter, seguretat.

Resumen

En el entorno educacional, el personal a cargo de la docencia de los estudiantes hace uso de forma frecuente de herramientas externas o de terceros. Normalmente, estos servicios proveen las funcionalidades deseadas, pero obvian algunos elementos claves.

Los dos problemas principales con esta práctica es la falta de seguridad y privacidad. En cuanto a la seguridad, los servicios previamente mencionados normalmente establecen conexiones con sus propios servidores, la cual cosa significa un intercambio de mensajes con un servidor potencialmente inseguro. Esto significa que los mensajes pueden traer adjuntos componentes maliciosos de las que el usuario no tiene conocimiento. En cuanto a la privacidad, no hay ningún tipo de garantía de que los servicios externos no almacenen datos del usuario, de que el servidor se haya visto comprometido con un ataque de intermediario o de que la red en uso no esté bajo el ataque de un *sniffer*.

En general, hacer uso de servicios externos significa que el usuario es susceptible a sufrir ciberataques. En base a este hecho, en este trabajo se presentará una aplicación que permitirá al usuario enviar y recibir datos de forma segura, local i basado en proximidad, incrementando de esta manera la seguridad.

Los datos escogidos a enviar en esta prueba de concepto son formularios, una herramienta sencilla per ampliamente usada en el ámbito educacional tanto por profesores como por alumnos, y la primera piedra sobre la que se pueden edificar nuevas funcionalidades como serian el seguimiento de asistencia o exámenes.

Esta aplicación se ha implementado con la API de Google Neaby connections y está escrita en dart a través del framework de flutter, con el objetivo de conseguir una solución multi plataforma. También se han estudiado otras tecnologías, que serán explicadas en detalle, definiendo que funcionalidades y que deficiencias presenta cada una.

La aplicación flutter resultante permite a los usuarios de Android hacer uso total de las funcionalidades definidas, aunque, hoy en día los usuarios de iOS no pueden hacer uso de las API de Nearby connections, limitando así su capacidad multi plataforma.

Palabras clave: aplicación, Google Nearby connections, multi plataforma, local, funcionalidades, tecnologías, flutter, seguridad.

Acknowledgments

I would like to express my gratitude to a series of persons and communities which contributed to my well-being and to my knowledge growth for the duration of this project.

Dr. Daniel Amo Filvà for his round the clock support and mentoring service. His encouragements and advices kept this project running while steering it as to not crash on a wall.

Mr. Alex Tarragó Pulvé for his disposal to attend my questions and for shedding light in some fields where I was in the dark.

Mss. Marta Zapatero i Sentís for cheering me up and providing emotional support, single-handedly pulling me out of some desperation moments.

Last, but not least, the communities of stack overflow and the official FlutterDev discord, for their educational interaction and discussions in helping to solve some problems.

Content

1	Introduction	1
1.1	Motivation.....	2
1.2	Theoretical framework.....	2
1.3	Problem	3
1.4	Objective	3
1.5	Methodology	4
1.6	Documentation layout	4
2	Research	5
2.1	Technologies considered.....	5
2.1.1	Classic Bluetooth	5
2.1.2	BLE	5
2.1.3	Wi-Fi	5
2.1.4	Others.....	7
2.1.5	Google Nearby.....	9
2.1.5.1	Nearby share	10
2.1.5.2	Nearby notifications.....	11
2.1.5.3	Nearby messages.....	11
2.1.5.4	Nearby connections	13
2.2	Technology conclusion	15
2.3	Plugins used.....	16
3	Development.....	18
3.1	Teacher module	19
3.1.1	Teacher view	22
3.1.2	Create forms.....	22
3.1.3	Endpoint list	24
3.1.4	My forms	26
3.1.5	Edit forms	27
3.1.6	Student management.....	28
3.1.6.1	List of answered users.....	28
3.1.6.2	List of answered forms	29
3.1.6.3	Building an answered form	30
3.2	Student module.....	31

3.2.1	Student view.....	35
3.2.2	Advertiser list	35
3.2.3	Building forms	37
3.3	Settings module.....	39
3.4	Common module.....	42
3.4.1	File management.....	42
3.4.2	Shared preferences	43
3.4.3	App Icons	44
3.5	Model module	45
3.5.1	Form	45
3.5.2	Endpoint data	47
3.5.3	Advertiser data	47
4	Conclusions	49
5	Limitations.....	50
5.1	Technological limitations	50
5.2	Development limitations.....	52
6	Future lines of work	53
7	Time estimation	55
8	References.....	56

Figure index

Figure 1 Global technological investment in education.....	1
Figure 2 Example of a student making use of a QR code in classroom.....	3
Figure 3 Bluetooth types according to SIG naming and their corresponding compatibilities	7
Figure 4 Some applications or projects who make use of a mesh network	9
Figure 5 Nearby by Google	9
Figure 6 Table showing the distribution of Android OS versions.....	10
Figure 7 Many to many communication model for nearby messages.....	13
Figure 8 Nearby connections P2P_STAR topology.....	14
Figure 9 Nearby connections workflow	18
Figure 10 Teacher activity diagram	19
Figure 11 Part 1 of the teacher's UML diagram	20
Figure 12 Part 2 of the teacher's UML diagram	21
Figure 13 Teacher main view	22
Figure 14 Form creation	23
Figure 15 Display of connected users while advertising the device	25
Figure 16 My forms view followed by the same view displaying an alert dialog	27
Figure 17 Edit form view	28
Figure 18 View listing all students who have stablished connection at least once	29
Figure 19 List of all forms the selected student has answered.....	30
Figure 20 Student activity diagram	31
Figure 21 Student modul UML diagram	33
Figure 22 Student main view.....	35
Figure 23 A modal bottom sheet displayed after selecting a discovered teacher.....	36
Figure 24 Change of colour to identify the teacher connected to.....	37
Figure 25 View displaying a built form received by the student.....	38
Figure 26 Settings activity diagram	39
Figure 27 Settings module UML diagram.....	40
Figure 28 Settings view	41
Figure 29 Common module UML diagram	42
Figure 30 Shared preferences .xml file as diplayed by Android Studio	44
Figure 31 Custom Teacher and Student icons.....	44
Figure 32 Model module UML diagram	45
Figure 33 Json of a form	46
Figure 34 Json of an EndpointData	47
Figure 35 Json of an AdvertiserData	48

Acronyms

AP: Access Point.

API: Application Programming Interface.

BLE: Bluetooth Low Emission.

BR: Basic Rate.

BSSID: Basic Service Set Identifier.

DNS: Domain Name System.

EDR: Enhanced Data Rate.

GATT: Generic Attribute Profile.

GFPS: Google Fast Pair Service.

HS: High Speed.

IEEE: Institute of Electrical and Electronics Engineers.

IP: Internet Protocol.

IR: Infrared.

ISM: radio spectrum reserved for Industrial, Scientific, and Medical purposes.

LAN: Local Area Network.

MAC: Media Access Control.

MVVM: Model View ViewModel.

mDNS: Multicast DNS.

OS: Operating System.

QR Code: Quick Response Code.

SIG: Special Interest Group.

SSID: Service Set Identifier.

TDLS: Tunneled Direct Link Setup.

TFG: *Trabajo Final de Grado*.

TTF: True Type Font file.

UUID: Universally Unique Identifier.

Bpoll

WebRTC: Web Real-Time Communication.

WPA: Wi-Fi Protected Access.

1 Introduction

Technology is present in every aspect of our lives. Its flexibility allows for its use in most fields while also offering an improvement on the already existing experience.

One field where technology is heavily present is in education. According to HolonIQ, investment in education has doubled since 2018 and has made a ten-fold increase since 2010, going from \$500m to \$16.1B[1]. This is best represented by Figure 1.

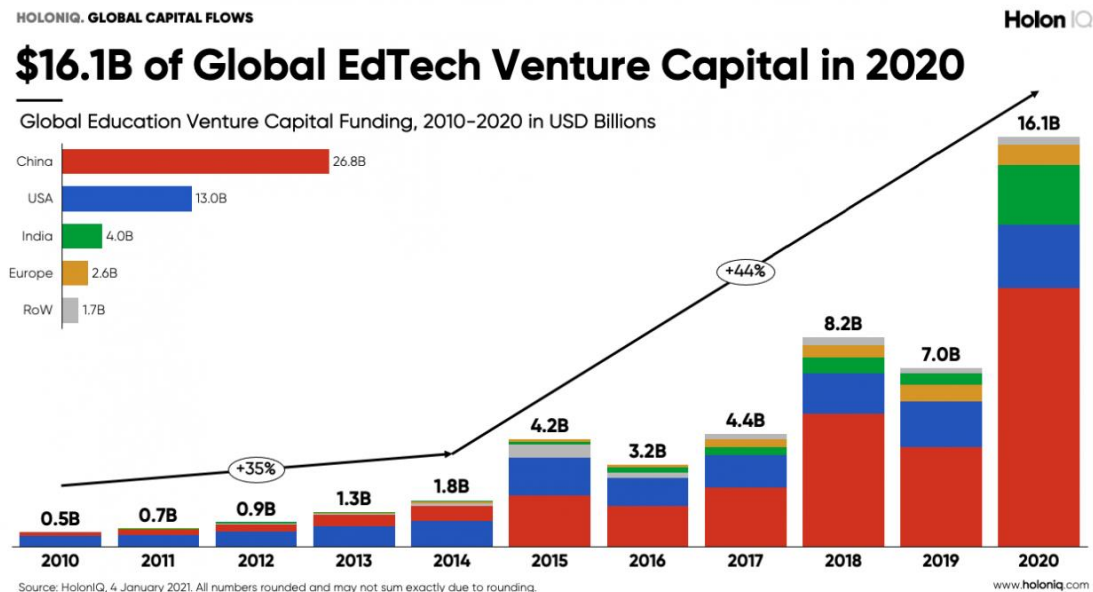


Figure 1 Global technological investment in education

Besides the heavy worldwide investment in education, taking online lessons, surveys, chats, and meetings have only been strengthened by the COVID-19 pandemic. Therefore, approaching technology from a simplistic use-on-feature view should no longer be applied.

Educational technology should not only be there to do something for its users, but to perform its duties whilst remaining save. Security and privacy can no longer be overlooked, but at times it is a characteristic that is left out.

This is most obvious when looking at figures for cyberattacks in 2020 compared to that of 2019. According to Bitdefender, ransomware attacks have increased seven-fold and IoT suspicious incidents have increased by 46%, amongst others[2].

On top of the increase in cyberattacks, with regards to education, schools are second in ransomware target, 87% of educational centres have suffered a successful cyberattack and 13% of all data security breaches from the first half of 2017 where related to the education sector, according to impact[3].

With a global cost of \$1 trillion in 2020[4], and from a range of 17 security components, a 60% increase over all 17 components over the last two years, “staying ahead of attackers is a constant battle and the cost is unsustainable”[5].

For these reasons, an application has been developed which allows for a teacher to share a form, and to receive his student's answers, all while staying off the net and using local communication.

Sharing forms on the net is an already existing feature brought by websites like Socrative¹ or Google forms² but the proposed application applies a coating of security brought both by Nearby's API and by the aforementioned offline local characteristic, following the guidelines given by the paper "Seven principles to foster privacy and security in educational tools: Local Educational Data Analytics"[6].

This solution also has the bonus that it does not require the user to log in or create an account, easing the user experience.

The resulting application can share forms in approximately a 100m radius and retrieve the results. This, however, is only available to Android users. Nearby's connections API is currently not available for iOS users. Other options that allowed for iOS compatibility were considered, such as Bluetooth or Wi-Fi Direct, but they had a series of shortcomings that made Nearby connections to be the final technology to be used.

1.1 Motivation

The widespread use of some of the websites that allows to create and share forms and being most of its public oblivious to privacy concerns, quickly lead to the realisation that this project presented an uncovered need, providing the motivation needed to work in this project. Besides the challenge set by the application development itself, this project also intends to raise some degree of awareness about the lack of measures taken in schools and colleges regarding user privacy.

1.2 Theoretical framework

This concept was born out of solutions that QR codes were able to solve, but quickly found its own limits. As explained by smekens³, QR codes is a great technology to share data on a 1 to 1 basis or a 1 to N unidirectional manner. Examples of this include having QR codes at the back of a book for more information or a fellow student review, having a QR code added to your foreign language homework that lets you listen correct pronunciations, or having QR codes that host a video of the latest class for those students who missed it (Figure 2).

¹ <https://www.socrative.com/>

² https://www.google.com/intl/en_en/forms/about/

³ <https://www.smekenseducation.com/communicate-with-qr-codes-in-th/>



Figure 2 Example of a student making use of a QR code in classroom

QR codes are great in this regard, but sharing information to several users is fine, but retrieving it from them can be quite a hassle. For reference, plickers⁴ is a great example of this. While many people use it and like it, it still has present some of the mentioned drawbacks, mainly, the fact that students need to have a printed card with the QR code in it⁵.

Having seen the QR code limitations, this project expands on this area by bringing forth the next technological step, wireless offline local communication between users.

1.3 Problem

The need for this application is born purely for two reasons:

- The lack of data security and user privacy when using external services providers in classrooms.
- The inability of the current local offline technology, QR codes, to send data to multiple users and to efficiently retrieve it. It is unidirectional in 1 to N communications.

1.4 Objective

To demonstrate that local data sharing is feasible in a local environment, by developing an application that allows teacher to share a form with his students in an offline environment. Students' answers should be received and stored in the teacher's device to be accessible later.

As a personal objective, in addition to the form sharing feature, to make it available to all users, regardless of what mobile OS they use, hence the development in the flutter framework.

⁴ <https://get.plickers.com/>

⁵ <https://www.youtube.com/watch?v=9VGF9dftSu4>

1.5 Methodology

As part of a one-member team, the methodology followed has been that of chaos model.

Chaos strategy follows a simple principle, to resolve always the most important issue, and by resolving it means to bring to a point of stability.

This was chosen for two reasons:

- Proof of concept application. The intended role of this application is to demonstrate that the proposed problem can be solved with the proposed solution. Additional software can optimize usability or design, but it is not strictly required to achieve the desired implementation.
- Time. Having no estimation of the costs in time for the development of any feature until it is hands on, means being unable to establish a schedule with which to guide the development. Spending time in fully implementing and covering a single feature might mean lacking it in the future when developing other necessary features for the project.

This approach made possible the usable development of all the basic features and only then the development for additional non-essential software.

1.6 Documentation layout

This document is distributed in two main blocks: research and development.

In the research block, an explanation is given regarding all the decisions taken that affected the latter development of the application. What technologies were looked up, their strengths and weaknesses, and why Nearby connections was the chosen one.

In the development block, an overview of the developed system will be given, followed by a detailed look to its four main modules. Inside each module, the .dart files used will be explored and detailed as to how they fit into the initial grand scheme of things.

After the detailed explanation of the decisions taken for the implementation and the implementation itself, the conclusions are exposed, followed by the limitations detailing the feeblest points of the application and future lines of work detailing a rework or fixes needed in the most critical aspects of the project, as well as what points should this project follow for future growth.

2 Research

In this chapter all the research done will be detailed. This includes a range of technologies that may fit into the local offline specifications of this project. An analysis of the technologies will be performed, stating strengths and weaknesses and in some cases, use cases of live applications will be provided.

2.1 Technologies considered

2.1.1 Classic Bluetooth

A technology developed in 1994, Bluetooth, or classic Bluetooth, refers to all Bluetooth versions up to 3.0. This includes Bluetooth 1.0 BR, Bluetooth 2.0 EDR and Bluetooth 3.0 HS. It has a speed of up to 24Mbps (in HS mode) and its compatibility is of no concern as all modern cell phones carry a Bluetooth antenna.

The major drawback of classic Bluetooth is that it can hold a maximum of 6-8 connections (depends on factors such as your hardware or amount of RAM). When the maximum is reached and a new device is connected, the oldest device is disconnected.

2.1.2 BLE

BLE refers to Bluetooth 4.0, released in 2010. It is “Low Emission” name comes from its distinctive feature that allows it to have the transmitter in an awaiting status most of the time, only activating when a connection is initiated. The connection and data transmission lasts only a few milliseconds, compared to the ~100ms it takes for classic Bluetooth (it has a longer pairing process) and it transmits data at about 1Mbps.

BLE is mainly used for wearable devices, smart IoT devices, fitness monitoring equipment and battery-powered accessories.

Although faster and 100 times more energy efficient, BLE was found not suitable mainly because the transmission between devices is limited to 20 theoretical devices (usually less when implemented).

Also, permission management in Android and iOS differ substantially, mostly because iOS generates its own UUID for each connected Bluetooth peripheral, whilst android uses MAC address and some characteristics and data types that may allow the device's identification[7].

Also, the advertisement data can be up to 31 bytes long, being composed of three flag bytes, a byte containing how long it is, one byte announcing the service data packet and 16 service UUID bytes, leaving 10 custom bytes which may not be enough.[8]

2.1.3 Wi-Fi

An alternative to Bluetooth is Wi-Fi. With Wi-Fi direct, devices can establish a direct connection between themselves and multiple devices at once. This feature is widely used when sharing internet from a device that has connection, to one that does not and is well documented[9]. But besides acting as a net provider, Wi-Fi direct can be used for data transfer.

While transferring files is not natively supported, there are several third-party software that manage it, such as sharedrop⁶ or Send anywhere⁷, so the goal for this project would be creating a similar application.

As opposed to Bluetooth, theoretical maximum user count is 254. Presuming the IP 192.168.0.0, the owner would be assigned the IP 192.168.0.1 and users would range from 192.168.0.2 to 192.168.0.254. Connectivity between devices should not be a problem as all devices have a Wi-Fi antenna, regardless of OS.

Although Wi-Fi direct looks like a promising candidate, it has several problems upon implementation which do not reflect the on-paper specifications mentioned earlier. While it is stated that it has a high maximum user count and compatibility, the reality shows that there are several connectivity issues between devices, especially if they belong to different manufacturers[10].

As a testament to the previous statement, a testing performed by Cirrent concluded that 20% of users fail when using Soft AP. They also state the fact that 40% of negative reviews in connected products are related to connectivity problems[11]. Reliability is highly regarded amongst users, so these figures demonstrate that this technology is not fit for this project's specifications.

A slight variation of Wi-Fi direct could be interconnecting devices within the same network, using only the local LAN.

In this case scenario, one device acts as the central AP through the local network, while the remaining devices establish a connection via the given 192.168.x.x IP. The IP assigned is random and it directs the user to 192.168.x.x:5000/index.html where shared files can be retrieved.

This functionality is preestablished in some devices and can be tested without the need to download any additional software. This method offers a theoretical maximum user count of 254, including the device acting as central AP (presuming the 192.168.0.0 IP, then users can range from 192.168.0.1 to 192.168.0.254).

While this solution is feasible, it is not entirely offline. Devices need to be connected to the same network for communication to be established and whilst the connection is not being sent to offsite servers, it still uses local resources which might be prone to attacks or already be compromised before its use.

Also, managing a html website hosted in the local network, via an application, can be quite a challenge as a filesystem should be implemented for the teacher to classify all his students/forms and users' permission should be tightened so that student cannot access another students' folder, avoid deleting submitted forms, editing submitted forms and so on.

⁶ <https://www.sharedrop.io/>

⁷ <https://play.google.com/store/apps/details?id=com.estmob.android.sendanywhere&hl=es>

Finally, all communications over Wi-Fi use WPA security, a technology widely used but quite insecure, as it is vulnerable to brute force attacks.

2.1.4 Others

As seen, Bluetooth has nice features which are hampered by the peculiarities of their versions but being that all Bluetooth signals work at 2.4Ghz ISM it is not out of the question to attempt and **hybrid solution** despite their differences[12].

Luckily, all mobile devices run on Bluetooth smart ready, a naming convention given by SIG to clarify that the Bluetooth being used ranges from 1.0 to 4.0, and that is both compatible with Bluetooth smart, comprised of BLE, and Bluetooth (or Bluetooth classic) comprised of Bluetooth 1.0, 2.0 and 3.0[13].

A compatibility chart between Bluetooth devices can be seen in Figure 3. As established by SIG:

- Bluetooth stands for standard or classic Bluetooth.
- Bluetooth smart stands for BLE.
- Bluetooth smart ready stands for a combination of Bluetooth and Bluetooth smart.










If your product bears this logo...	It's compatible with products bearing any of these logos...
	  
	 
	

Figure 3 Bluetooth types according to SIG naming and their corresponding compatibilities

To communicate devices, both technologies mostly share the same profiles specifications. Profiles are a set of rules that define what task will this Bluetooth device perform[14]. To communicate both a BLE device and a classic Bluetooth one, they must be using the same profile. Among other profiles, GATT is an example or a compatible one, which allows data exchange.

Although and hybrid solution can be taken to term, its implementation provides a challenge for a developer with basic Bluetooth knowledge. Also, it may improve overall performance, but

it will not overcome the shortcomings of both classic Bluetooth and BLE, therefore it is deemed an unfit approach.

Another approach could be using **IR communication**, but this technology was quickly found out to fall short of all minimum requirements for this project. Namely, it does not have a very long reach (~30m), it can be obstructed by objects such as walls, smoke, modern devices have dropped the use of IR blasters and most importantly, with IR you can only connect to one device at the time[15].

Ultrasound opens another window of opportunities. The device's speakers can be used to emit a set of signals, inaudible for adults (~17+kHz), that transmit data to other devices which receive it through their microphone[16].

While this is a valid data transferring method, its shortcomings come quite obviously, as in an environment with a lot of ambient sound, this technology is unusable, not to talk about the fact that its transmission speed is around 7kbps and that it has no pairing system, meaning any device within range can listen and read the data transmitted.

Mesh network. Many projects have observed similar problems when choosing a technology for local data sharing, and the solution for them was to create a mesh with all the devices that wished to connect. This way, mixing previously discussed technologies such as Bluetooth and Wi-Fi direct, all devices can be interconnected.

A clear example of this would be **Firechat**, which although is currently discontinued, it reportedly was used in several protest such as Honk Kong protests (2014), Ecuadorian protests (2015) and Catalan protests (2015).

Also, another mesh is that of **Bridgefy**⁸, where devices share messages until each one reaches its destination. In our use case, roles would have to be carefully determined as the role of "teacher" would be thinly separated from the role of "student".

Luckily enough, Bridgefy does have a Github repository⁹ with android samples, however, this is not the case for flutter, where there is a complete lack of documentation and plugins.

Another mesh example, the **Serval**¹⁰ project also offers a mesh built up using a combination of Wi-Fi and Bluetooth. Serval does offer a free license to any user wishing to use its software, however, in a similar fashion to Bridgefy, its deployment in flutter is non-existent.

As a final example, **p2pkit**¹¹ also offers connectivity between two nearby devices. Much like some of the previously detailed examples, it uses a mix of Bluetooth and Wi-Fi to achieve local communication. While it has an API, it is not available for flutter and it comes at a cost (free version available too, but quite limited).

⁸ <https://bridgefy.me/>

⁹ <https://github.com/bridgefy/bridgefy-android-samples>

¹⁰ <http://www.servalproject.org/>

¹¹ <http://p2pkit.io/>

Besides each individual project API, the problem with these approaches lies in the fact that programming a mesh requires a substantial effort by an experienced team of developers, so it is out of scope for this project to develop a mesh for flutter using any of the previously mentioned services.



Figure 4 Some applications or projects who make use of a mesh network

2.1.5 Google Nearby

An API presented in 2017, google nearby offers users an API which makes usage of Wi-Fi, Bluetooth, BLE, IP and audio. The aim of combining these technologies is to ease the communication between proximity devices[17].

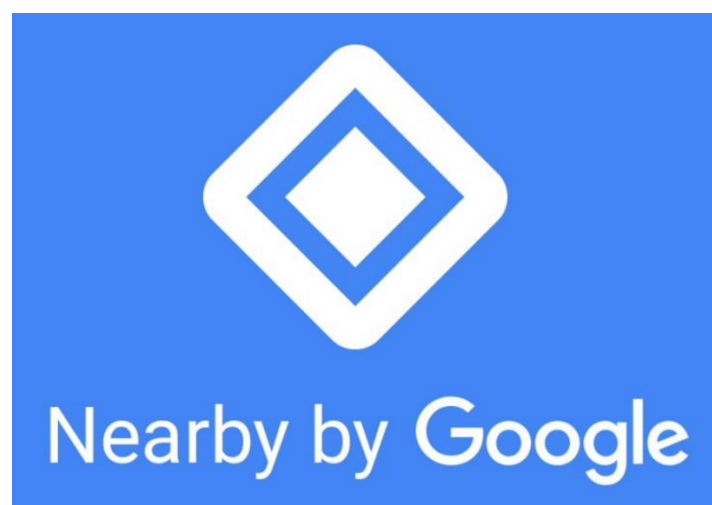


Figure 5 Nearby by Google

Despite having five modules, only four stands today, having nearby notifications discontinued for various reasons. In addition to this fact, nearby share does not have an available API nor documentations to develop on it.

In spite of this, a brief description will be given for all members of the family, excluding nearby pair, which is oriented to the pairing of nearby devices such as speakers, headphones and car kits via BLE and GFPS.

2.1.5.1 Nearby share

Presented in 2019 under the name “Fast share” and released in august 2020 under the name “Nearby share”, it intends to be the android equivalent to the popular AirDrop, available for iOS only.

Nearby share will make use of Bluetooth, BLE, WebRTC and peer-to-peer Wi-Fi to both be fully offline and to make use of the most appropriate and faster technology at that given time. Also, nearby share is capable of sharing images, files, links, and other contents and allows the user to share data anonymously or not. It also offers a visibility setting that makes you visible to contacts only, to stay hidden, or visible to only some of the contacts[18][19].

Available for devices running Android 6+, this means it is available for 84.9% of the devices, according to Android Studio, as shown in Figure 6.

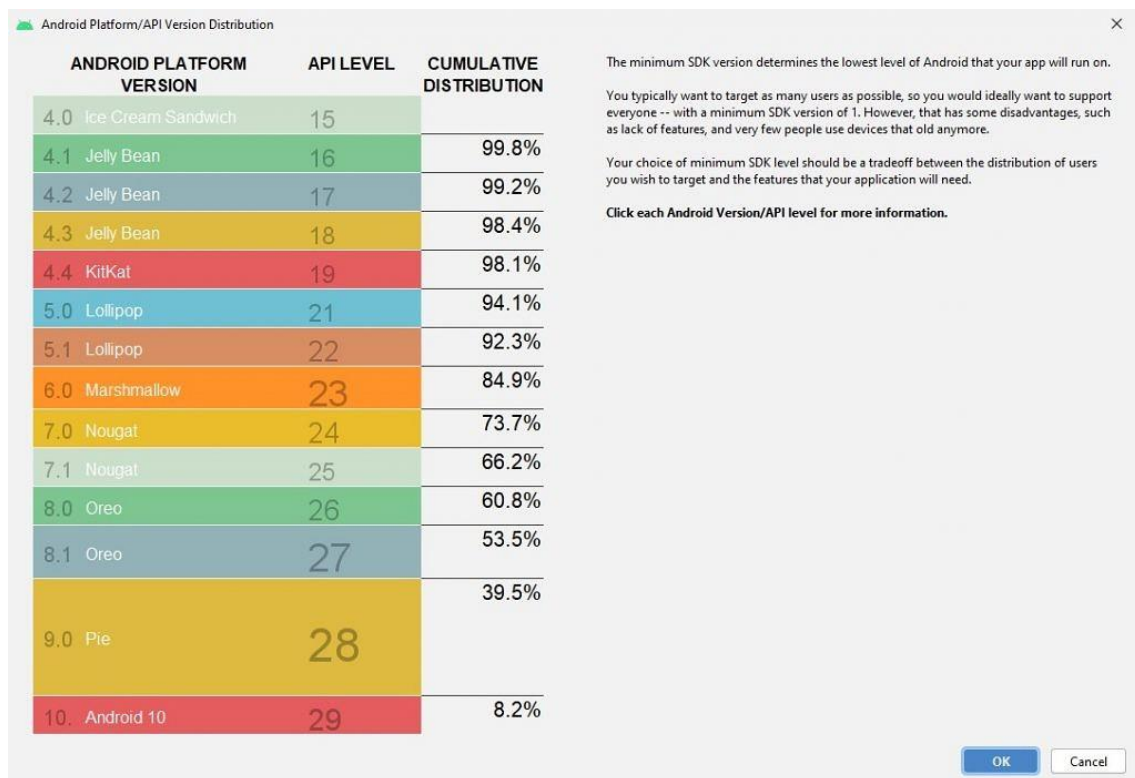


Figure 6 Table showing the distribution of Android OS versions.

While it may seem a good candidate, there are some nuisances we must point out as to why this option was discarded.

As a starter, this is a service provided by google, not an API. That means there are no docs, no codes samples, all there is, is an update to your phone and this feature appearing on it afterwards. At the most, Microsoft offers an API that manages shared connections, mostly to integrate them into the Windows 10 OS and share data between devices[20] as a part of the Project Rome (Microsoft's cross-device experiences platform for apps).

While Microsoft has its own project for cross-device experiences, other services might not, and google share is, as of now, not supported for platforms such as iOS and Macs. They “plan to try to expand the feature to additional platforms in the future”. Not surprising as it is rather new. Also, it must be said that testing revealed that sharing data to more than one device at the time proved impossible.

Finally, regarding flutter packages, since there is no API available, there is also no packages available to code this service into a flutter application.

2.1.5.2 Nearby notifications

Developed in 2015, and part of the API proper of Google’s nearby technology, nearby notifications allows Android users to see a message associated to an app or website which is location-specific based. As stated in the official documentation, amongst other examples, it can “launch conversations or chatbots inside messaging apps”[21].

On top of that, it can open an application to display its contents, or if the user does not have it, direct him to the corresponding download page. This could greatly improve user experience and ease use for young pupils, elder pupils, or students who are not tech savvy.

As a downside to notifications is that it transports URL data through BLE, with all that BLE implies. In this case, presuming you make use of Eddystone beacon, from the 17 bytes available, you are left with 11 bytes for payload. On top of that, the payload is a URL, so it needs internet access to retrieve its contents. Also, the channels used are unidirectional making a student answer unviable[22].

Finally, and most importantly, google deprecated this API in 2018, just three years after its release. This decision was taken because the use of nearby notifications led to an increase of spam for the users. This spam could be filtered or tuned but nonetheless, google stated that this solution, and therefore this technology, did not live up to their high-quality standards, this removing support. The beacons are still available through the Proximity Beacons API¹², which in turn was recently deprecated too on December 7th, having its support formerly shut down on April 1st, 2021[23].

2.1.5.3 Nearby messages

Next in the nearby family is messages. This technology uses a mixture of Bluetooth, BLE, Wi-Fi and near ultra-sonic audio to communicate near devices. Having this many technologies running under the hood, means that nearby messages can leverage the strengths from each one, to supplement the remaining weaknesses[24].

¹² <https://developers.google.com/beacons/proximity/guides>

Additionally, the inclusion of ultra-sonic audio allows for the developer to adjust the range of the local communication. By default, it is set at about 30 meters, but when in ultra-sonic mode, it can be reduced to 1.5 meters[25].

Also, this API is not deprecated and does not even require a Google Account. It is not heavily payload limited and is available both for Android and iOS.

As an example of its usage in a local environment is card against humanity test application, called Manatee, developed by Shepherd, Meredith C. for his thesis. While she demonstrates that a local application through this technology is viable, she also gives some insights regarding flutter development:

“Having no previous Android development experience, I had planned to use Flutter, Google’s newly introduced cross-platform app development framework, in the hopes that it would provide an easier road to incorporating Nearby Messages. However, despite the fact that both projects were produced by the same company, no Flutter plugin for Nearby existed at the time, nor did one become available until March of 2019. (Cross, 2019) In the meantime, my advisor suggested Xamarin.Forms, a more established cross-platform framework with a far more robust mechanism for incorporating platform-specific code. With Flutter’s Nearby Messages support lagging behind Xamarin’s, and with Flutter still in Beta at the time (Sneath, 2018), there was no good reason not to switch to a more mature product.”[26]

While the previous statement is only a year old, it is to be believed that it still stands as of today and defeats the personal objective imposed for this project, to use the Flutter framework.

Also, nearby messages makes use of a many to many communication model (Figure 7), meaning that much like Wi-Fi direct, permissions as to who the “host” of the session is must be established, ensuring to student can peek at another student’s answers.

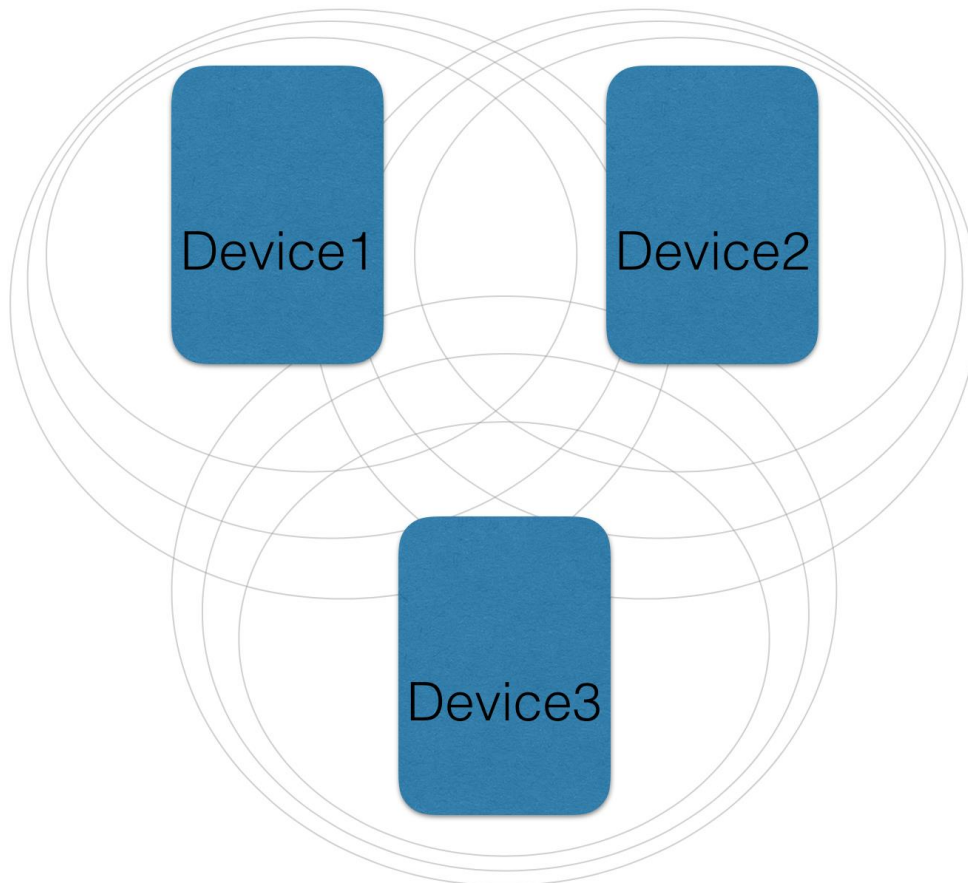


Figure 7 Many to many communication model for nearby messages.

Besides development hardships, nearby messages states that it is a nearby based communication technology, which is true, but it also requires internet connection (albeit, not all devices must be in the same network). This requirement becomes an apparent issue as nearby messages only share locally a token, used to retrieve a message stored in Google's servers. While only used that have been locally shared this token may access the message, its necessity to connect to an external server defeats the purpose of this project, thus rendering this technology unfit for use.

2.1.5.4 Nearby connections

As the last member of the nearby family, introduced in early 2015, nearby connections offers similar features to that of nearby messages, but with some key differences. The main difference, which came alongside nearby connections 2.0 (released in 2017), is the capability on functioning in a fully offline manner[27][28].

In more detail, nearby connection uses BLE, Bluetooth and Wi-Fi hotspots to advertise, discover and connect to peer devices in a fully offline manner. On top of that, connections are encrypted, low latency and high bandwidth since, much like nearby messages, it leverages the strengths of each technology to supplement its weaknesses.

Notice that, unlike nearby messages, nearby connections does not make use of ultra-sounds, thus its unable to adjust its default range of 100 meters[29].

Nearby connections also includes three connections topologies:

- P2P_STAR 1:N
- P2P_CLUSTER (much like the one defined in Figure 7 for nearby messages) N:M
- P2P_POINT_TO_POINT.

This fits perfectly into our project as the P2P_STAR topology, see Figure 8, can be utilised by the teacher as the central node, thus eliminating the need to control data transmissions amongst devices if using a mesh such as P2P_CLUSTER. P2P_POINT_TO_POINT has not been considered at all.

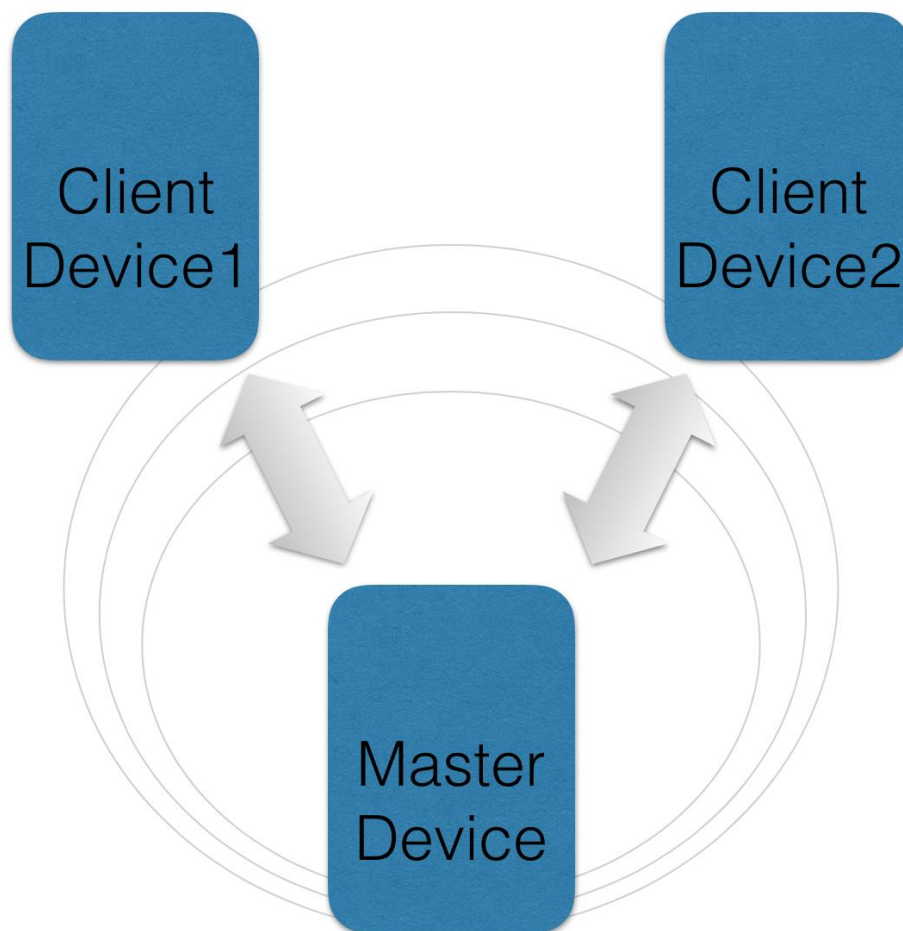


Figure 8 Nearby connections P2P_STAR topology.

While there are multiple topologies, not all of them use the combination of BLE, Bluetooth and Wi-Fi hotspots mentioned before. The limiting factor is the Wi-Fi hotspot, which can only host or connect at any given time. This means that for the P2P_STAR and P2P_POINT_TO_POINT topologies, BLE, Bluetooth and Wi-Fi hotspots are used, but for the P2P_CLUSTER topology, only Bluetooth and BLE is used, and as we have seen, the max number of devices connected via Bluetooth is a theoretical 7-8, with a real 3-4.

This does not mean that P2P_STAR and P2P_POINT_TO_POINT have an unlimited number of available connections. They use Wi-Fi hotspots, true, but the Wi-Fi hotspots can hold up to 10 connections, improving over Bluetooth and BLE, but not by much[30].

On top of the topology, there are three means through which the encrypted data can be transferred. Bytes, file, and stream.

- Bytes: Byte arrays with a maximum length of 32k.
- File: Files of any size.
- Stream: Data generated on the fly, used mostly to transfer audio or video without knowing the end beforehand.

Much like other technologies discussed, it has some shortcomings too. Mainly, it is not compatible with iOS devices. It is a feature Google is planning to introduce in a future, but there is no estimation as to when they might implement iOS compatibility.

Also, both nearby connections and nearby messages are not allowed to work in the background, they must always be in the foreground[31]. This is explicitly designed by Google so that the final user is always aware when his device might be advertising, sharing, or discovering.

This design has been implemented to prevent the user from involuntarily allowing nearby to perform actions on the background, much like when users forget Wi-Fi on Bluetooth is ON in their devices, and also it is to avoid an excessive battery loss. Using Bluetooth, BLE and Wi-Fi means a lot of battery is consumed when performing actions. Therefore, having the activity in the foreground serves the dual purpose of letting the user know when nearby is working, and also passively indicate to the user when he should stop nearby, resulting in reduced battery consumption.

2.2 Technology conclusion

Having explored so many technologies for local data transport, an informed decision can be taken. Whilst there are means through which you may operate in a local offline manner, there is always a shortcoming such as compatibility or number of connected devices.

There are paid APIs that offer all the functionalities and features required for this project but being a proof of concept for local offline data transmission in an educational environment, making use of paid services is out of the question.

As such, the chosen technology for the project is **nearby connections**. While it may not be compatible with iOS devices, it offers features such as P2P_STAR topology and a choice in data transmission types. Also, although it has a low maximum device connection cap, it is still greater than other technologies. Additionally, it is compatible with a wide range of devices, it has several plugins available for flutter and Google plans to add iOS support in the future, meaning a development in flutter today can be the iOS application of tomorrow.

2.3 Plugins used

For the development of this project, several plugins were used. This, and other plugins, may be found in the official pub.dev¹³ website. Some of the plugins used in this project are not used to its full extent, or are not used at all, for they are there as a next step in the project. For more information about next steps, see the chapter Future lines of work.

As of now, the plugins implemented to its full usage are:

- **cupertino_icons**: It is the default package included in any new Flutter project. It contains all default icon assets.
- **nearby_connections**: A package that offers a neat usage of Google's nearby connections. Developed by Prerak Mann and in collaboration with Gourav Saini, it is available both from the packages' website and in the form of a Github project. As an example of its usage, it was implemented in a Monopoly Money Handler¹⁴ project.
- **shared_preferences**: This package allows for simple platform-specific data storage. The data is stored in a pair of key-value pair. This plugin will be further explained in the chapter Shared preferences. Its developers are the official flutter.dev group.
- **path_provider**: A plugin used to navigate the filesystem of the device, such as the temp and the data directories (amongst others). Much like shared_preferences, its developers are the official flutter.dev group and more information regarding its usage can be found in the chapter File management.
- **device_info**: Also developed by the official flutter.dev group, this package offers information from the device such as model, name, UUID... This package is used throughout the project, although shortly, it is used to obtain a unique reference from each connecting device as connection id changes each session.
- **flutter_launcher_icons**: This last plugin, developed by the fluttercommunity.dev group, simply offers the means to update the application's launch logo.

Regarding the unused, or partially implemented plugins:

- **wifi_iot**: This plugin, developed by alternadom.com, offers means through which Wi-Fi connections can be handled. It checks the Wi-Fi status, it enables/disables Wi-Fi, gets information such as SSID, BSSID, frequency, signal strength...
This plugin is used in the settings window, detailed in the chapter 3.3 Settings module and it is partially implemented as some of its features have been overtaken by nearby connections API, as it offers the means to automatically turn on the Wi-Fi in case it is turned off, rendering this plugin obsolete. Despite this, it is kept in the project as it may be used for future features.
- **image_picker**: This official plugin (flutter.dev), allows the developer to pick images from the existing library and take pictures with the camera. It was implemented into the project as the means to test the capabilities to send images with nearby connections.

¹³ <https://pub.dev/packages/>

¹⁴ https://github.com/mannprerak2/monopoly_money_game

As images are successfully sent and received, this plugin was kept in the project as a future feature might encompass the listing of connected students to the teacher by name and picture, making it easier for the teacher to recognize the connected users. As of now, this feature is of low priority hence it is not implemented.

3 Development

For the development of this project, the workflow of nearby connections must be understood first. As nearby connections is the main feature of this project, most other features will need to adapt to its workflow, hence its importance.

Nearby connections works in an advertise and discovery manner, meaning that for a connection to take place, at least one device must be advertising at the same time for another device to be discovering. On top of that, we stated previously that Google enforced the use of nearby connections while the application is in the foreground, and also recommends notifying the user of the current advertise or discovery mode via an icon or a special screen[32].

Taking this into account, the application will have a workflow time utilised only by the establishment of the connection amongst devices. As specified in Figure 9, this encompasses the discovery/advertising phase, the connection request by the discoverer, and the acceptance of the connection on both ends. Having to send a request and then an acceptance for the connection, means the student will have to interact with the application twice.

From this point onwards, the connection is established, and data can be sent, as well as other actions can be performed.

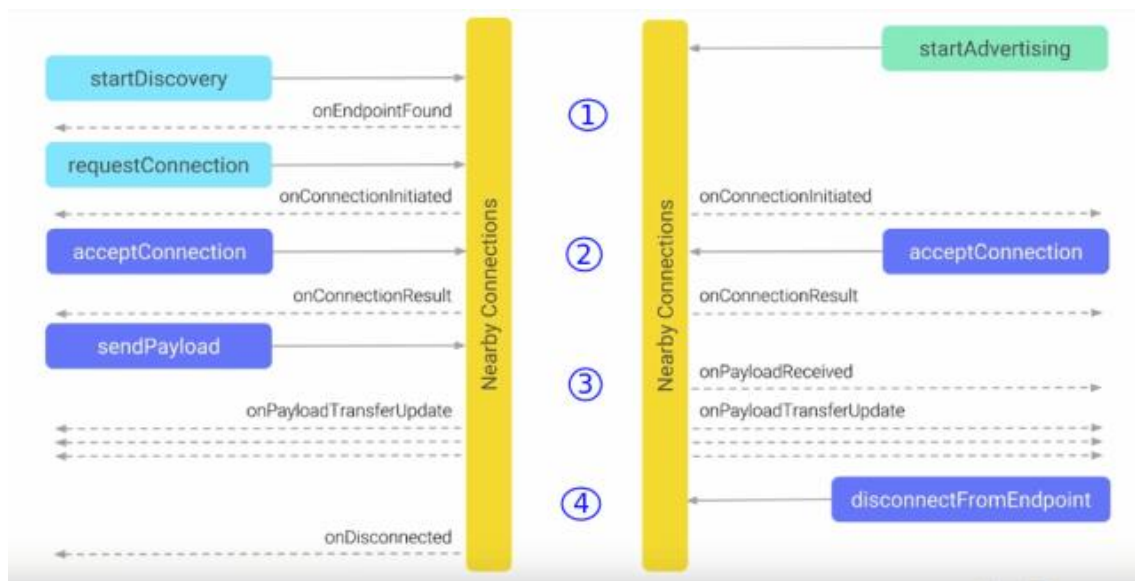


Figure 9 Nearby connections workflow

Other actions, which may also be used before the connection is established include tinkering with the settings, and specific to the teacher, create forms, edit forms, and consulting previously sent forms by the students.

What follows is a detailed explanation for each module used in this project.

3.1 Teacher module

This module implements all the functionalities available to the teacher. As an educational application, the teacher is the one in control of most features, therefore this module contains most of the features implemented for this project and thus is the one with most importance.

The features, visualised in Figure 10, encompass:

- Creating forms.
- Editing/sharing/deleting forms.
- Consulting connected devices.
- Advertising/halt the advertisement of the device.
- Disconnecting all connected devices.
- Consulting the student's answers.

These features are distributed amongst several .dart files, grouped according to their nature and stage in the nearby connections' workflow. The individual .dart file are detailed in the following chapters, although a general diagram for the teacher module can be found in the following page (Figure 11 and Figure 12).

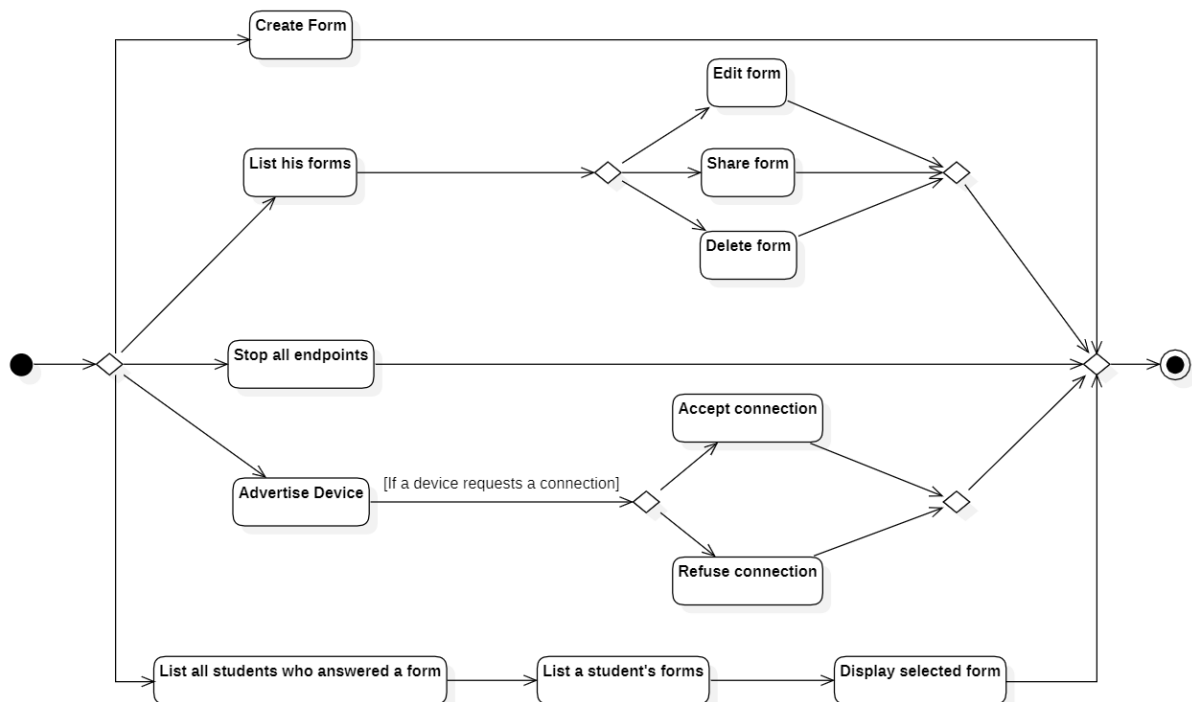


Figure 10 Teacher activity diagram

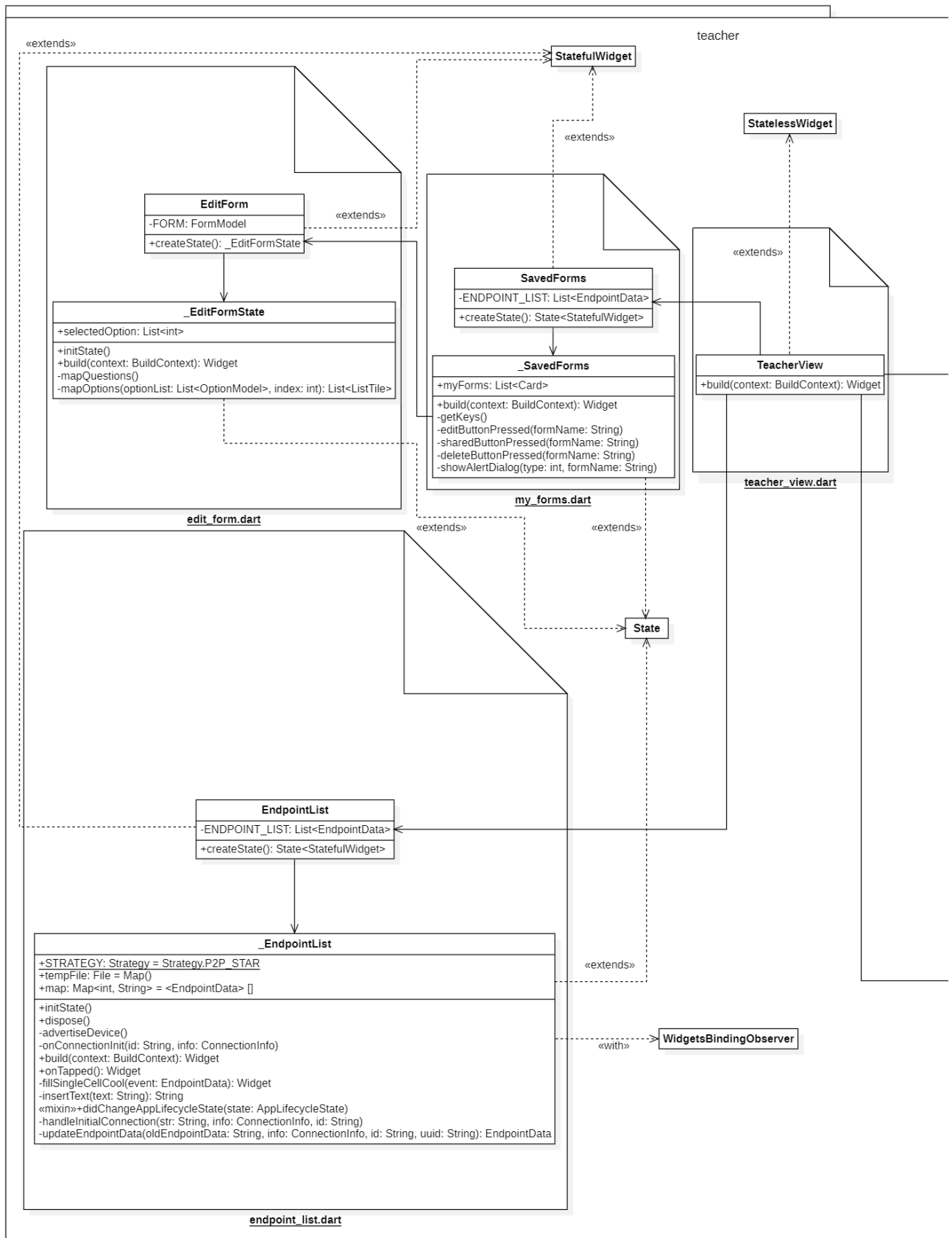


Figure 11 Part 1 of the teacher's UML diagram

3.1.1 Teacher view

This simple view serves as a main menu for the teacher, see Figure 13. Easily available, he may tap in any of the four buttons to navigate to their corresponding window. This view extends from StatelessWidget, meaning its state does not change on runtime. It is the only class that has this characteristic.

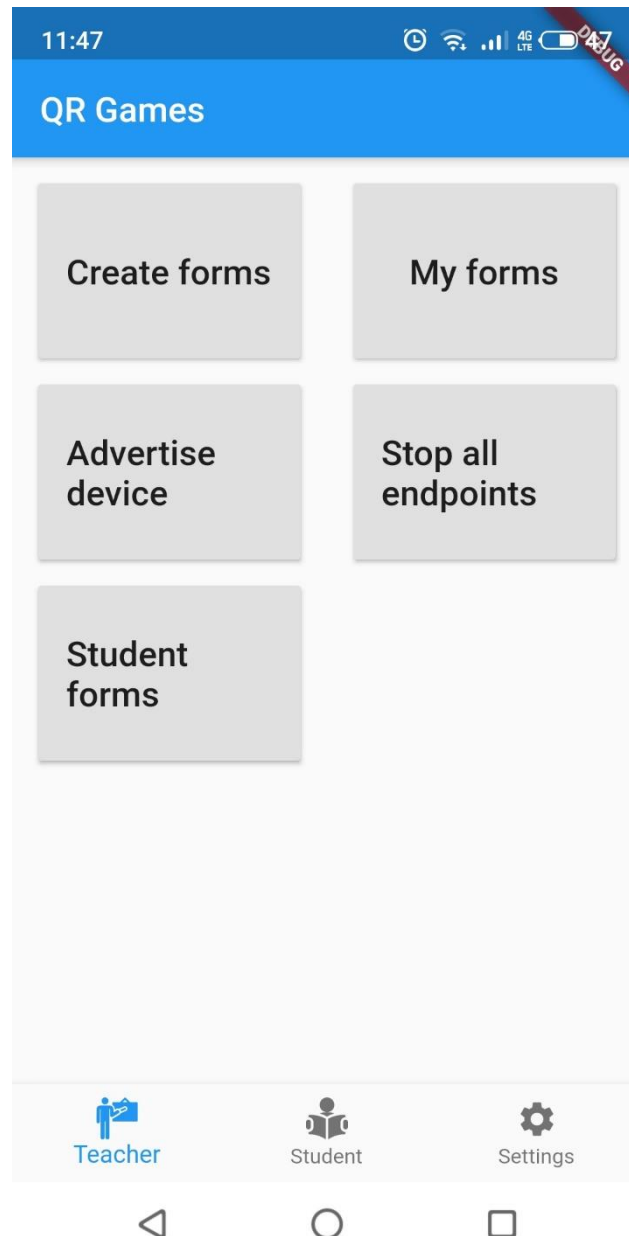


Figure 13 Teacher main view

3.1.2 Create forms

This view, designed to dynamically create forms and map them to its corresponding model, might be one of the most complex of the project.

When creating a form, a user might add as many questions as desired, and each question may contain as many options as desired too. The user also has the option to remove any of the inserted questions and options, although when it comes to the options, the user can only remove the last option added. This decision was taken as a measure to keep a clean view.

Implementing the possibility to delete any option is feasible, but it would require a button, or similar, for each option, on top of the already existing marker that each question has, making the view too crowded. The end result is a view that displays a neat template easy to navigate (Figure 14).

The screenshot shows a mobile application interface for creating a form. At the top, there is a status bar with the time 20:38, signal strength, 4G LTE, and battery level at 53%. Below the status bar, the title "Form title" is displayed. The main content area contains two question blocks. Each block has a "Question title" and a "1. Option". Below each option, there are "+" and "-" buttons. A trash icon is located to the right of each question title. At the bottom of the screen, there is a navigation bar with a hamburger menu icon and a "+" button. Below the navigation bar, there are two buttons: "Cancel" and "Create form".

Figure 14 Form creation

Therefore, within the create forms feature, we find the Option class and the Question class. That brings to total of classes used to three, and each of the three classes (CreateForms,

Question and Option) extend from StatefulWidget, with each having their own corresponding class that extends their parent's state (_CreateForms, _Option and _Question).

As such, upon user interaction, questions and answer are created without limits. Once the user is satisfied with the form created, and has given it a title, he can save it into the device. To do such a thing, each question with its own corresponding options is iterated through to create a Json file reflecting the form. It is this form the one it is stored in the device using shared preferences (refer to chapter 3.4.2 Shared preferences for more information on shared preferences).

3.1.3 Endpoint list

To develop this view, a close look has been taken at Google's guidelines where they state that the user should always be fully aware of when the device is in discovery/advertising mode and that the user should always know what information is being shared.

While information sharing will come at a later stage, this window serves the purpose of advertising the device to the students. When the "Advertise device" option is selected, the device enters advertise mode and displays in a list all the connected devices (empty if none are connected). As long as the teacher remains in this view, his device is being advertised and all connections requests are prompted. Loosing focus of the application or returning to the previous view (via return button or back arrow in the view's title) will cause the device to stop advertising until the focus is returned to the advertise view.

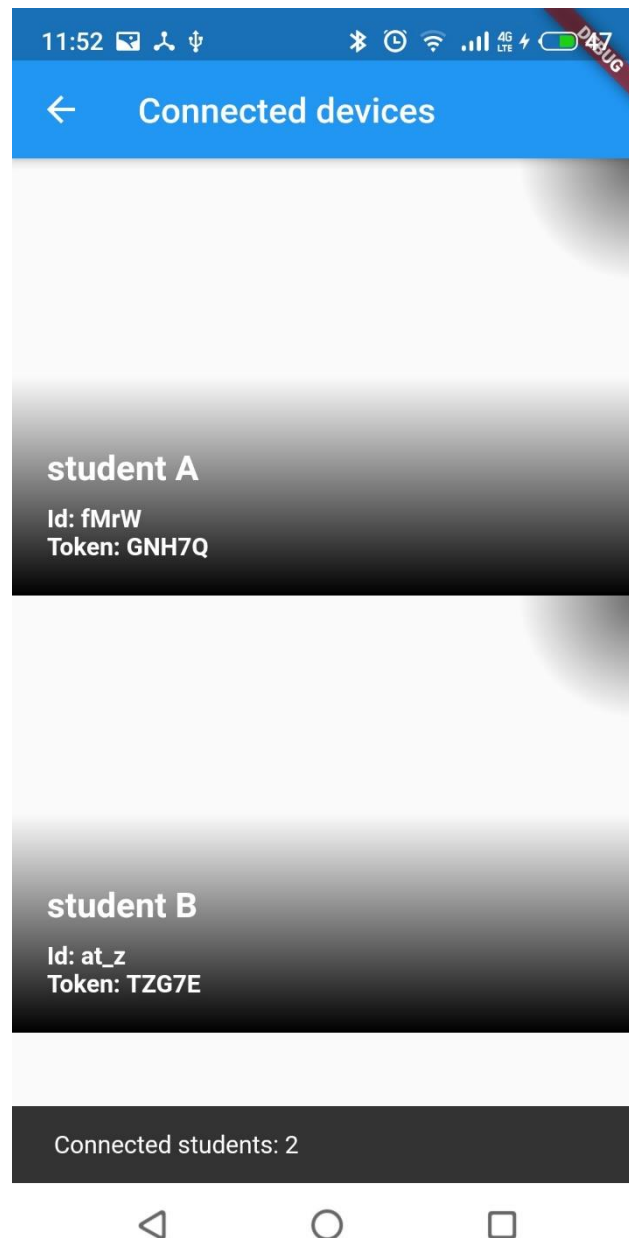


Figure 15 Display of connected users while advertising the device

It must be said that the official documentation states that connections requests can be received after stopping the advertise mode, as discovering users may see an advertising device (while it is advertising) and make a later request for a connection once the advertising has ended. In other words, devices may recognise each other during advertise/discover phase, but it is the user who requests a connection, which can be done after a device has been recognised.




Upon accepting connections, the list fills up with the connected devices, which display the users' name, the connection ID, and the verification token for that connection. Also, a space for an image has been added. This image is intended as a future feature, which is detailed in chapter 6 Future lines of work.

The user's UUID is not displayed. Instead, when first meeting a new user, it is stored into shared preferences alongside the user's name. Upon reconnection, the UUID is checked to see if it matches that of any previous user connected. If it does, and the user is under another name, then the user's folder is renamed with the latest name. Besides checking if the user had previously connected to the device, it also checks whether the connecting user is already connected, thus avoiding duplicates in the list.

The usage of UUID (via the `device_info` plugin), is done as a consequence of nearby connections not implementing the means to identify previously connected devices. It makes use of the ID and validation token, but those are valid only for the duration of the current connection. A device connected under the ID "sampleID" might connect later with the ID "anotherSampleID", thus making ID not reliable.

3.1.4 My forms

In order to consult your forms, this view lists them for you. Within each displayed form, three actions can be taken:

- Delete form  : Deletes the form from memory. It includes a confirmation pop up via an alert dialog.
- Edit form  : Allows you to make changes to an existing form.
- Share form  : It sends the form to all connected users. To be able to send the form, it converts it to Json (after retrieving it from memory) and sends it as a byte payload. Much like the delete form functionality, it displays a confirmation pop up via an alert dialog.

While basic, this view offers the users some flexibility in the management of the forms. Besides this, it is the only view in the app that allows for a share of information, and listing all available forms makes it easy for the user to decide what to send and when to send it, keeping in line with Google's quality standards.

On top of being the only view in the project where information can be sent, this view sports alert dialogs for both sharing a form and for deleting it, as can be seen in the following Figure 16.

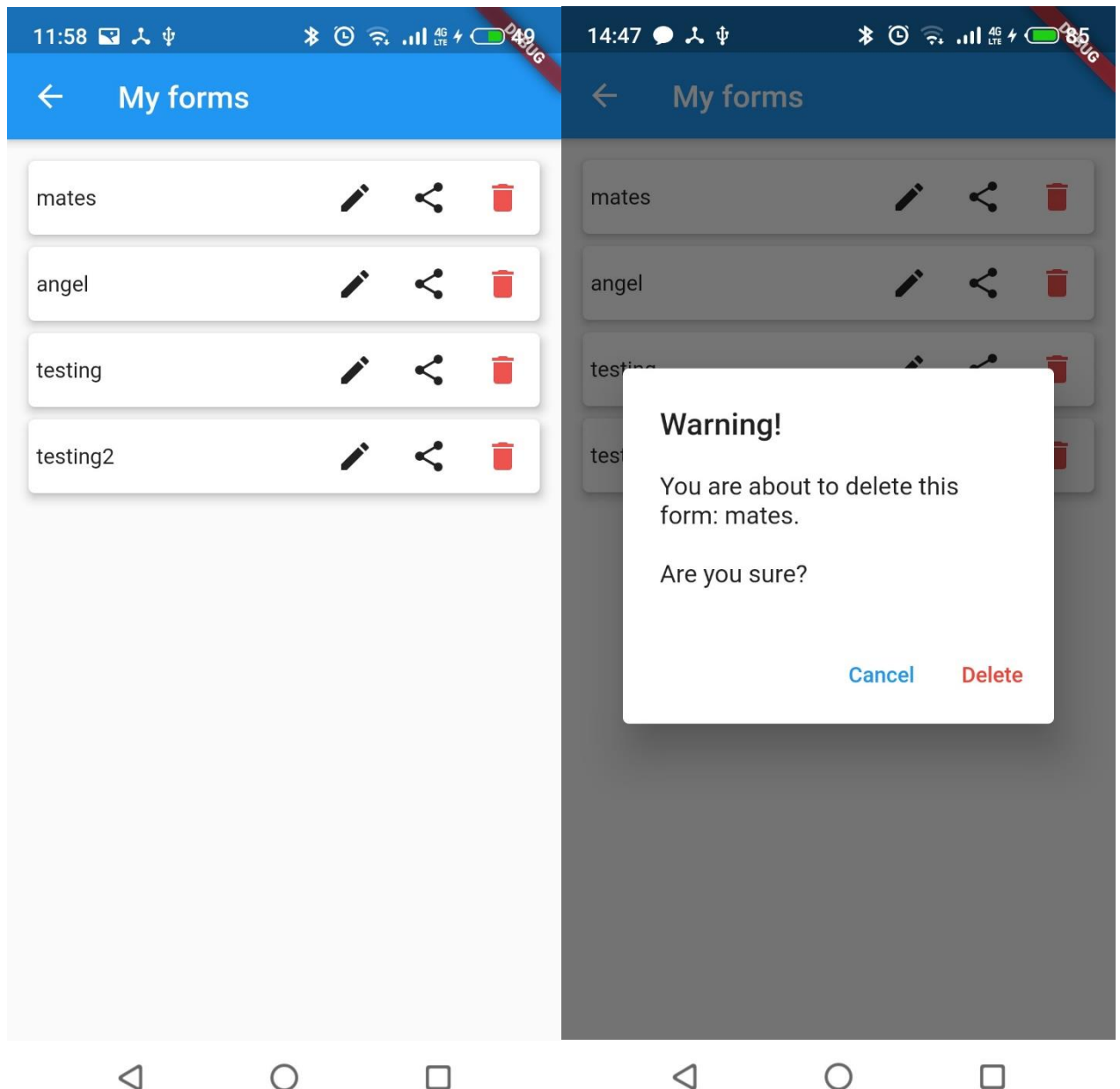


Figure 16 My forms view followed by the same view displaying an alert dialog

3.1.5 Edit forms

When selecting the “Edit form” option mentioned in chapter 3.1.4 My forms the user enters editing mode (Figure 17).

In this mode, the user can modify any question or option contained in a form. After modifying the desired fields, the user can either discard the changes by pressing the back button or navigating back with the AppBar arrow or might save the changes with the floating button displaying the “Save” icon.

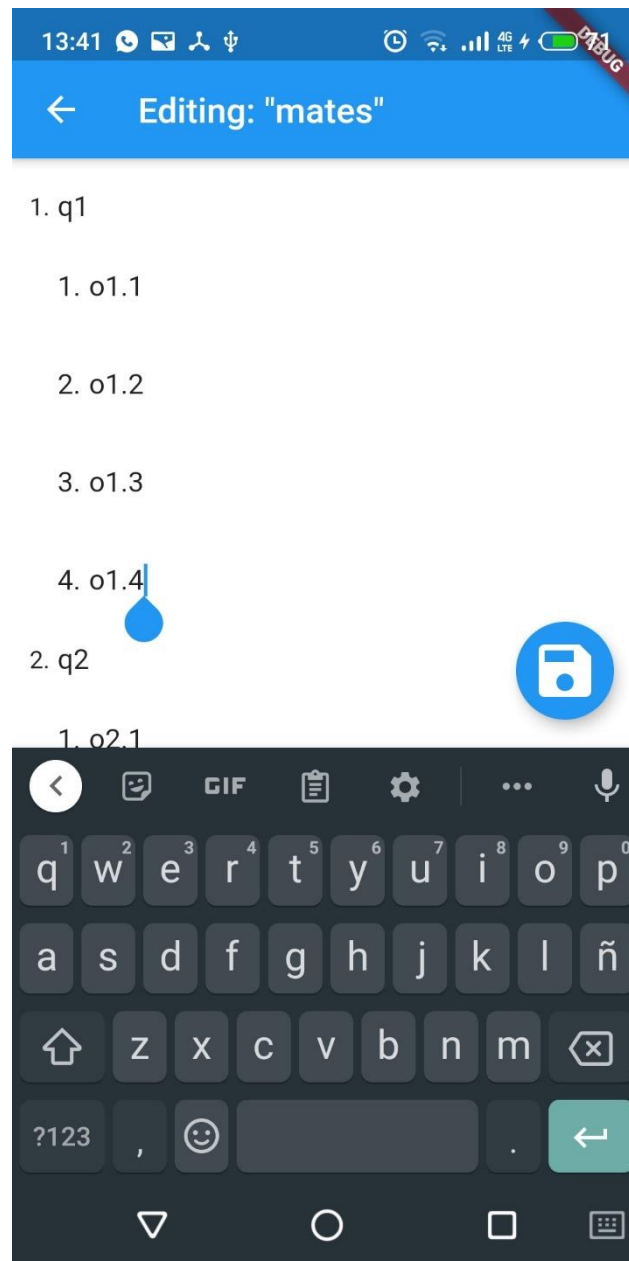


Figure 17 Edit form view

3.1.6 Student management

This sub directory is designed to enclose all actions the teacher can take with a received form from a student. These actions, so far, include only to display the answered form, although the steps previous to displaying a form include showing the filesystem where all student's answers are stored (by student name) and within each student, a list of all his/her forms.

All modules contained within this directory make use of the functionalities associated with file management, which can be found in more detail in chapter 3.4.1 File management.

3.1.6.1 List of answered users

When receiving a connection, regardless of it is a new connection or one from an existing user, the filesystem is updated, or created, with a directory with the user's name. It is on this filesystem that the student's response will be stored.

When accessing this view, the teacher will be able to see a directory for each connected user (bear in mind that a connected user may not necessarily have sent any answer to the teacher).

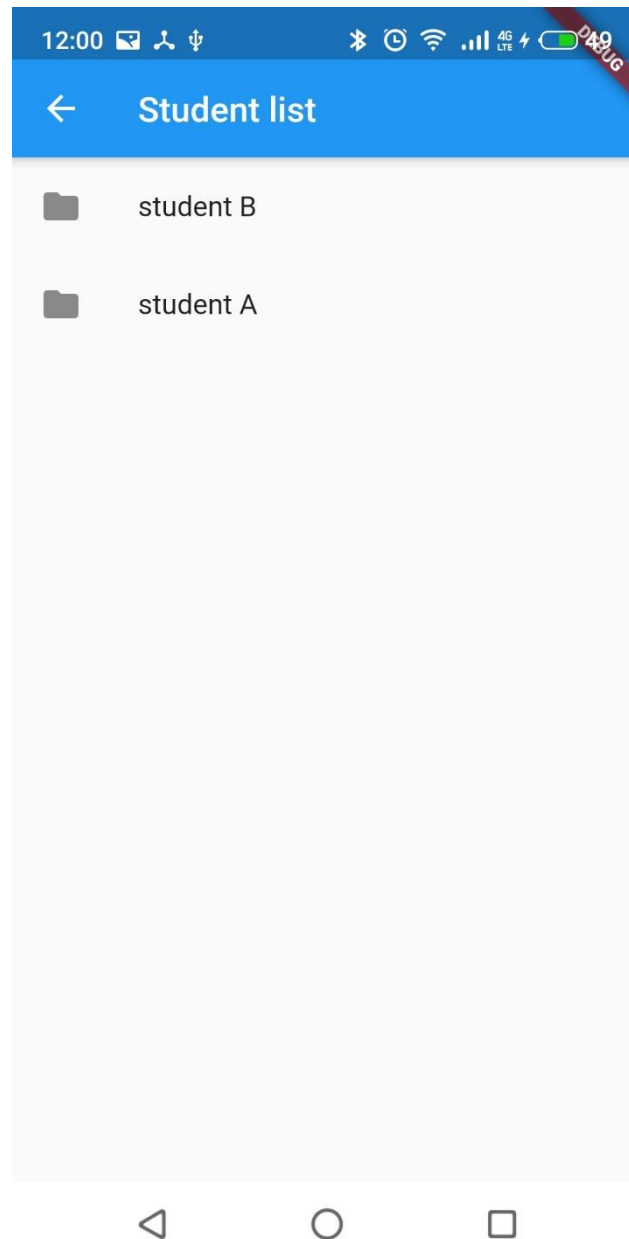


Figure 18 View listing all students who have stablished connection at least once

3.1.6.2 List of answered forms

In a similar fashion to the previous point, within the list of users when accessing a user that has responded to the teacher, a list of all his responses is displayed. The forms answered by the student are saved using the same name under which they were created and sent to the student. Selecting any form will display it.

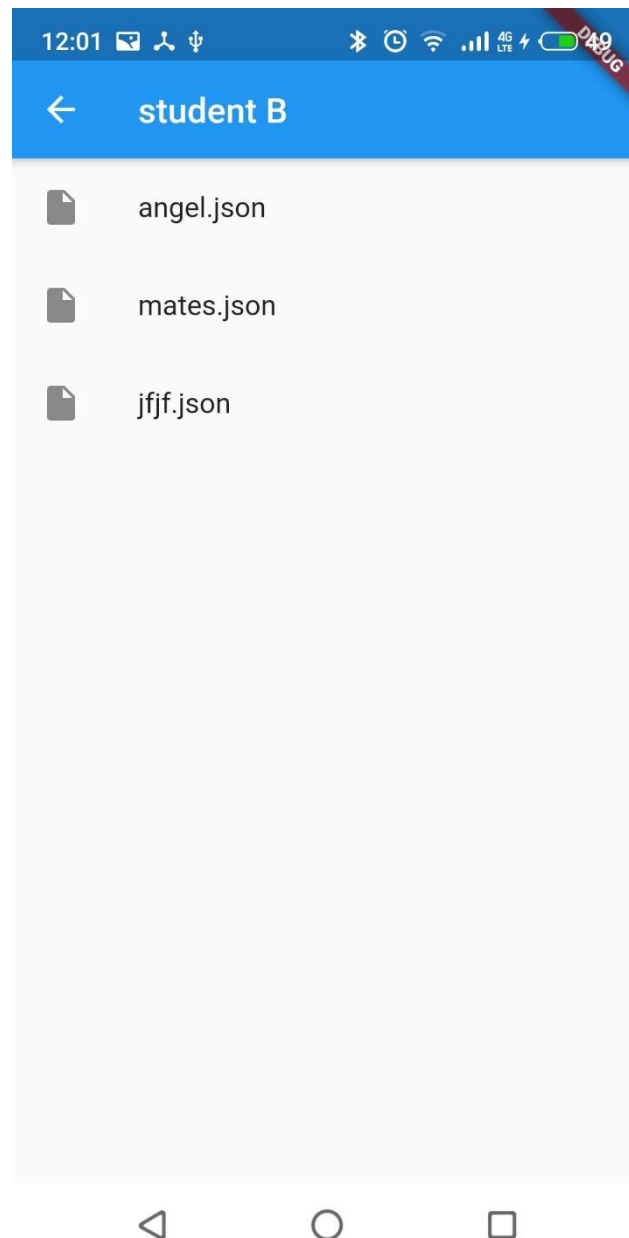


Figure 19 List of all forms the selected student has answered

3.1.6.3 Building an answered form

As the last step, selecting a form from a student, will begin the process to display it to the teacher. This process accesses the saved form stored in a file in the system, parses the retrieved form in Json format and dynamically creates a view with uneditable radio buttons to indicate the user's choices.

3.2 Student module

The student module holds all the necessary logic to complement the teacher in the workflow of this application. It can discover devices, receive forms, display them, fill them and send them back to the original sender.

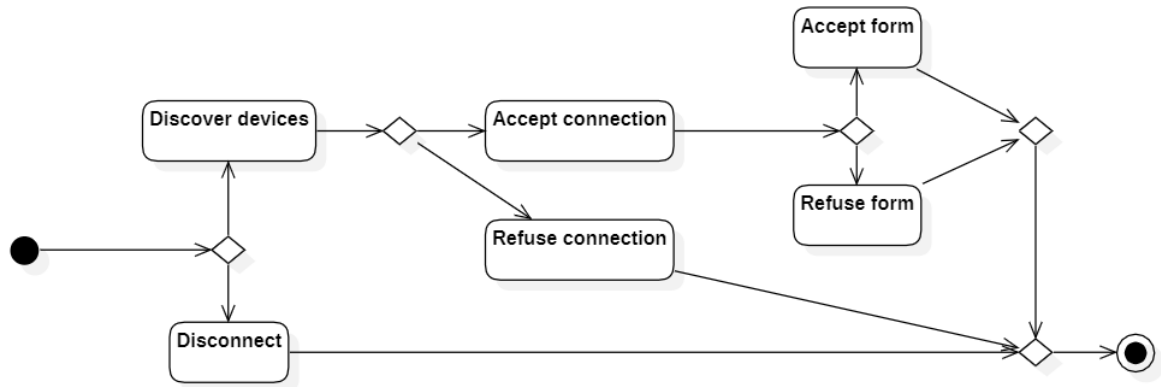


Figure 20 Student activity diagram

It currently does not hold as many features as the teacher, due to two reasons. On one side, this is due to the nature of the project. The teacher, just like in classrooms, holds the brunt of the work that needs to be done, so he is the one in charge of the creation, communication and management of the forms, remembering users, displaying filled forms... while the student has a more laid-back role and must simply discover, connect, read forms and answer them.

The other reason is that features such as a historic of forms, or saved teachers as "contacts" held a low priority in this project and are not in place yet. Having the student logic contained within its own module, eases future development.

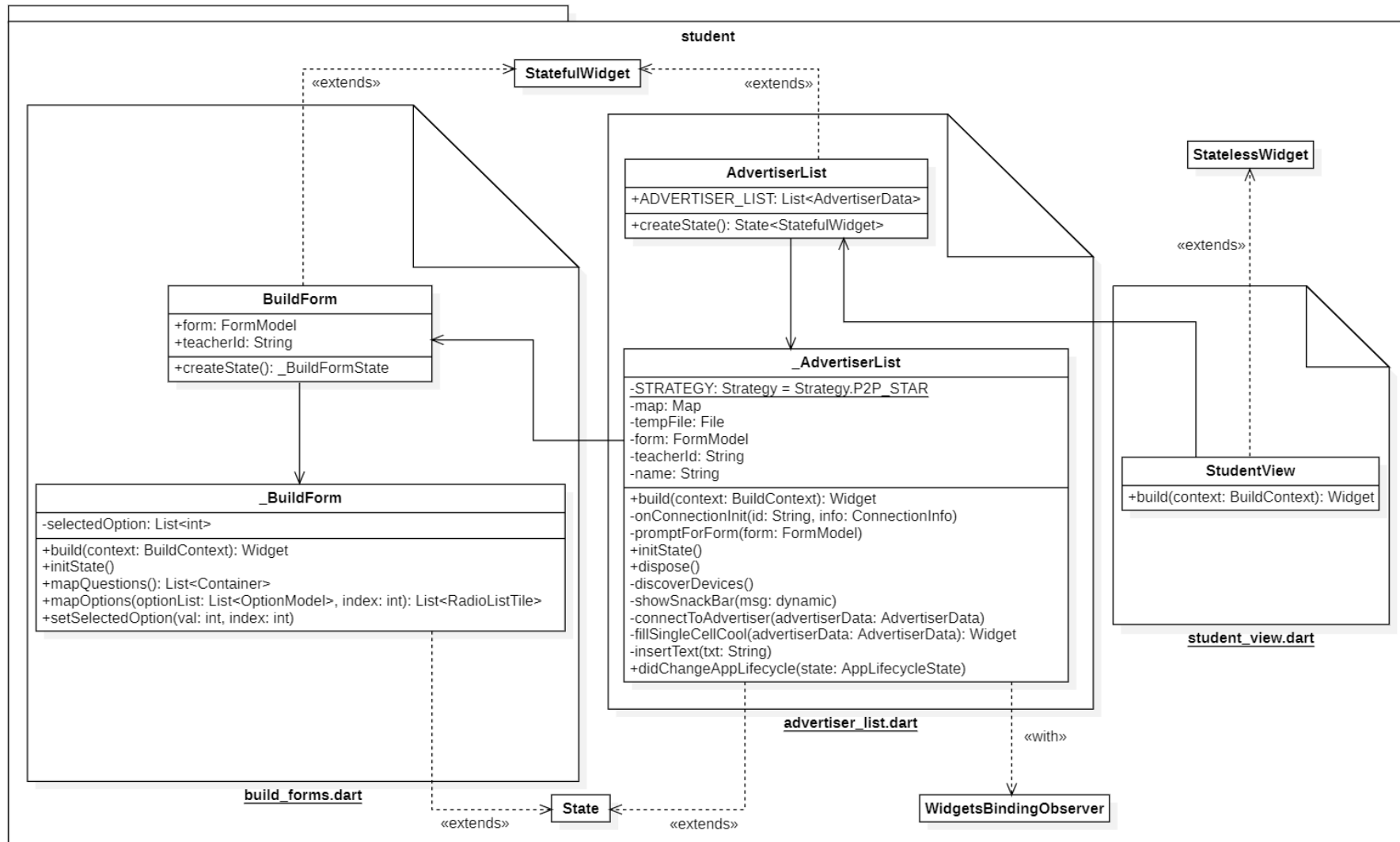


Figure 21 Student modul UML diagram

3.2.1 Student view

In a similar fashion to the teacher's main view, this view display simple buttons through which the student can navigate to the available options. This class extends from StatelessWidget, meaning its state does not change in runtime and is the only class in this module that does so.

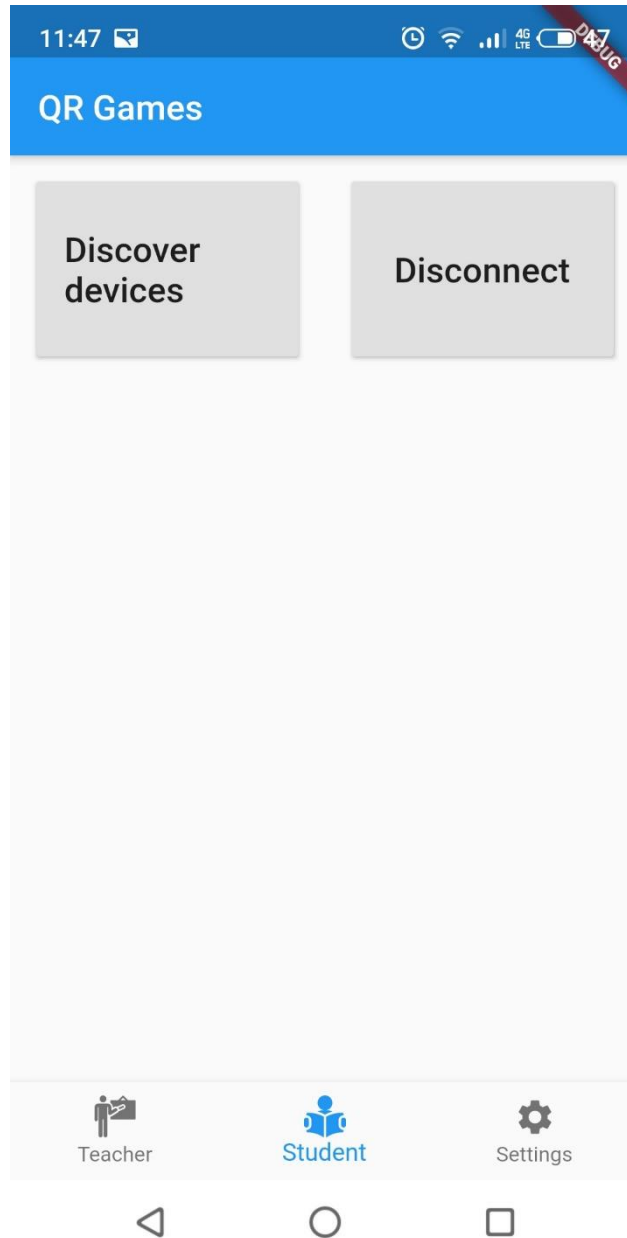


Figure 22 Student main view

3.2.2 Advertiser list

Taking into account Google's best practices for nearby connections[32], displaying this screen will also trigger the `Nearby().startDiscovery` method. This method will discover any nearby devices and display them in a list. The user must select the desired endpoint to which request a connection. The request for a connection is prompted by a modal bottom sheet (Figure 23).

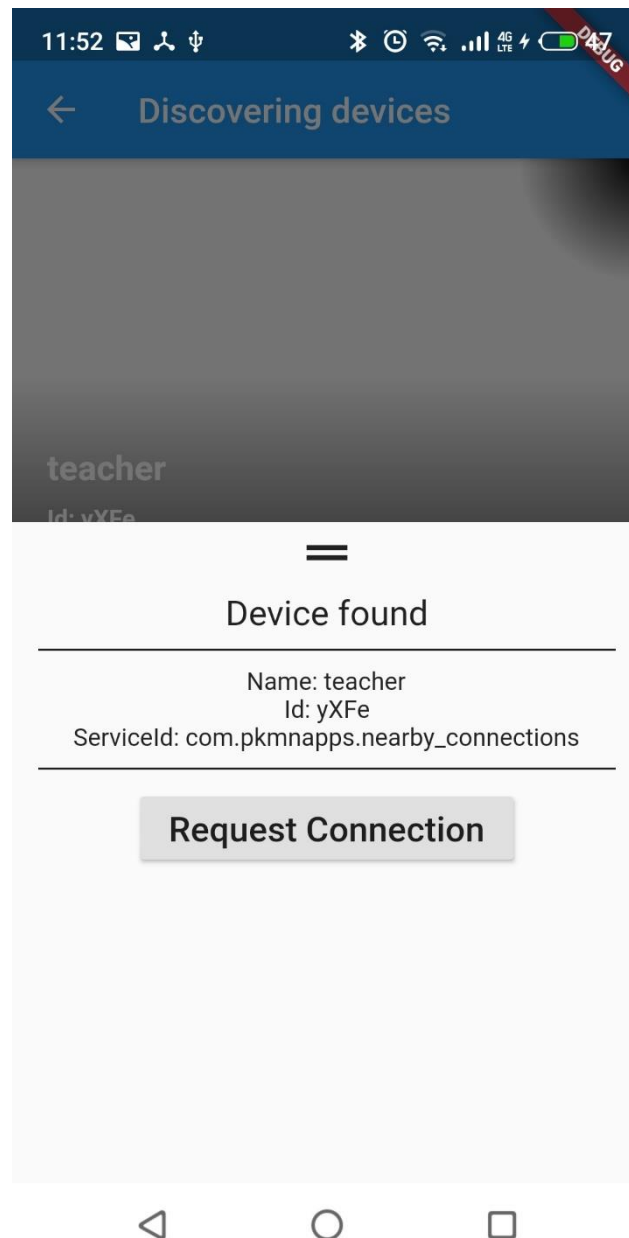


Figure 23 A modal bottom sheet displayed after selecting a discovered teacher

Upon connection to the teacher, the view modifies the teacher to which the connection has been established by changing the colour to a green tone while maintaining non connected teachers in a black tone. This has been done so that the student can easily identify to which teacher (presuming there are several in the same environment) he has connected to (Figure 24).

Besides starting to discover when navigating to this view, this view makes use of the `WidgetsBindingObserver` in order to detect when the application is focused or unfocused and act accordingly. Navigating out of this window via the return button or the top navigation return button will also trigger the device to stop discovering.

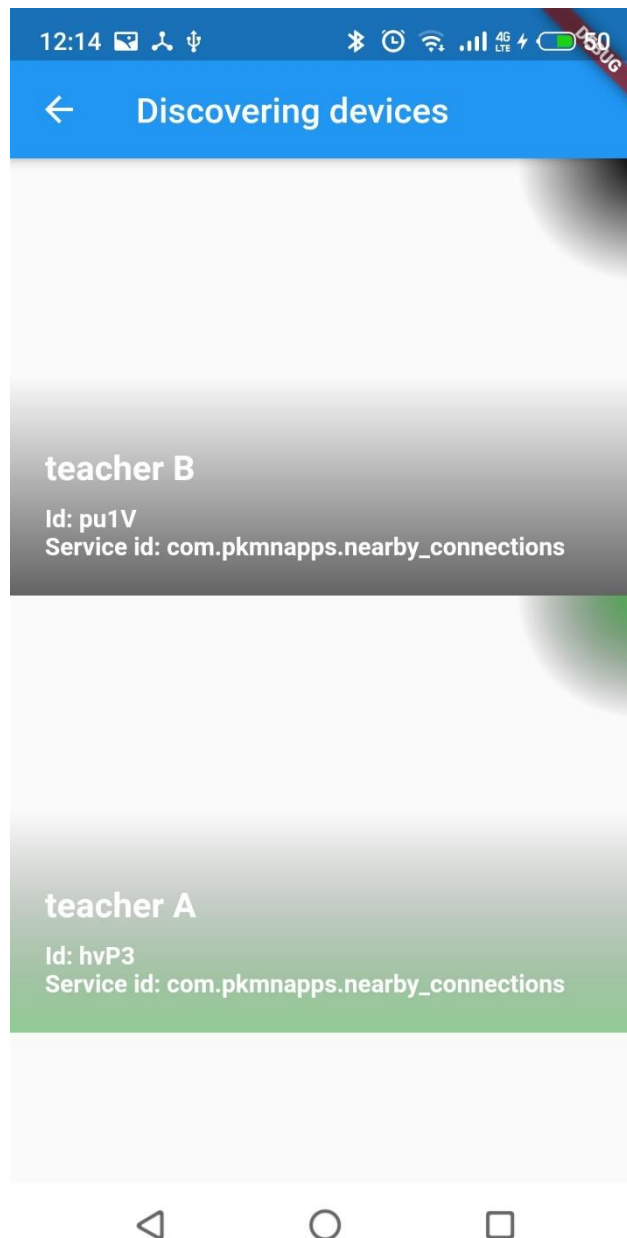


Figure 24 Change of colour to identify the teacher connected to

3.2.3 Building forms

Upon receiving a form from a teacher, it needs to be parsed and displayed properly. Building forms is the class in charge of displaying the parsed form. Parsing it is done by the previous class, advertiser list.

Upon receiving a form parsed into the model class `FormModel` (see chapter 3.5.1 Form), building forms iterates over all the fields contained in the form. For each field, it creates a corresponding widget such as a text for the question title or a radio button for each question, see Figure 25.

The screenshot shows a mobile application interface for a poll. At the top, there is a status bar with the time 16:31, signal strength, and battery level. Below the status bar is a blue header with a back arrow and the name 'angel'. The main content area displays three questions, each with a group of radio button options. Question 1 (q1) has two options: o1.1 and o2.1, with o2.1 selected. Question 2 (q2) has two options: o2.1 and o2.2, with o2.1 selected. Question 3 (q3) has two options: o3.1 and o3.2, with o3.1 selected. A 'Submit' button is located at the bottom of the form. The bottom of the screen shows the standard Android navigation bar with back, home, and recent apps icons.

16:31

← angel

1. q1

☐ 1. o1.1

☒ 2. o2.1

☐ 3. o3.1

2. q2

☒ 1. o2.1

☐ 2. o2.2

3. q3

☐ 1. o3.1

☐ 2. o3.2

Submit

Figure 25 View displaying a built form received by the student

The radio buttons corresponding to each question are mapped into a radio list tile, a grouping logic that allows for a radio button of the same group to be deselected when another in the same group is selected. Also, the radio buttons can be toggled to be deselected in case the student wishes to not answer a question.

3.3 Settings module

The settings window allows the user to:

- Check the state of the device's Wi-Fi
- Check the state of the device's location.
- Flip the status of these two technologies (if it is on, turn it off and vice versa).
- Change his name.

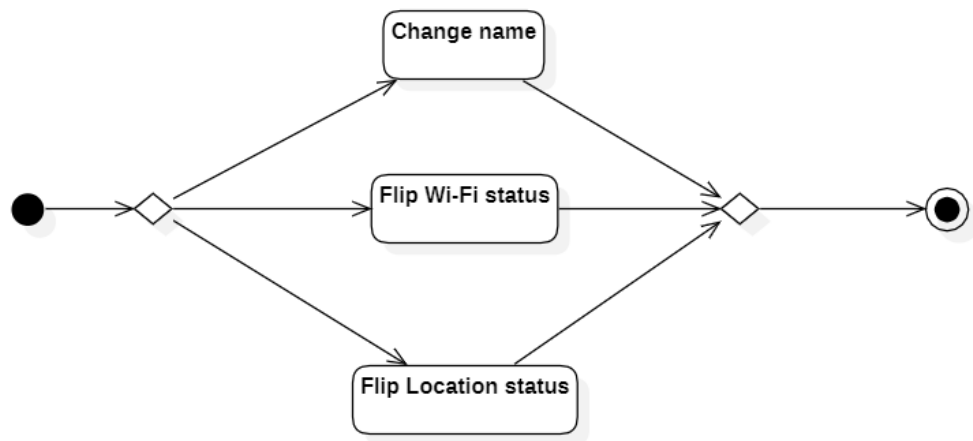


Figure 26 Settings activity diagram

The name will be stored in the shared preferences file when changing tabs, unfocusing the application and when pressing the submit button in the keyboard. This name will also be the name under which the directories in the teacher's endpoint will be created and will be the name displayed to other devices when engaged in discovering/adverting mode.

The view is formed by a stateful class with it is corresponding state class, as seen in Figure 27, and it makes use of the `WidgetsBindingObserver` mixin[33] to detect when the app is unfocused, and act accordingly.

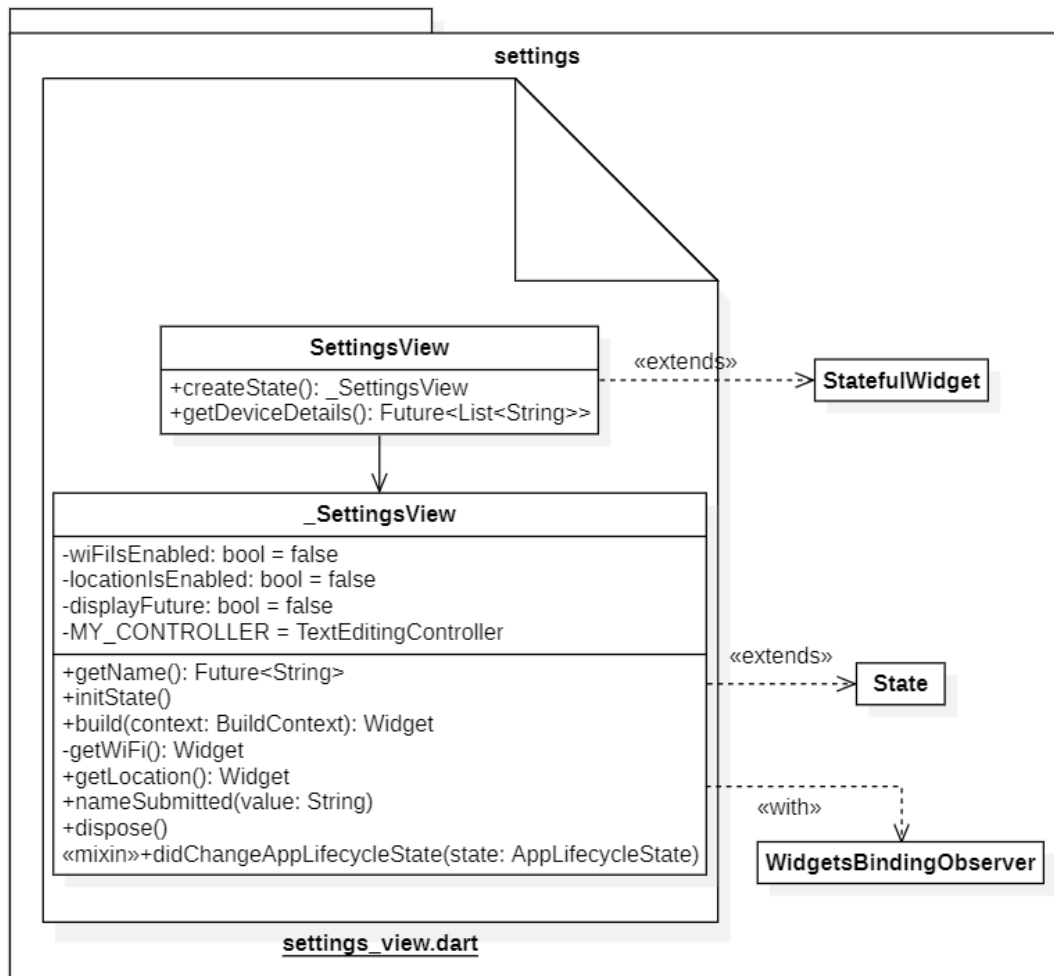


Figure 27 Settings module UML diagram

This view (Figure 28) offers the key functionality of naming the device. Other functionalities are not as critical as they are also embedded inside the nearby connections API. They have been added to the view as a mean to improve usability by giving the user an in-application view of the current state of the device.

Besides improving usability, the sole existence of a settings view opens up a great deal of possibilities which may not be strictly necessary for this proof of concept but will definitely be necessary for a release.

Some features may include enabling night mode, editing the default folder for the student's directory, choosing language, adding the device's Bluetooth status and so on. For more details, refer to chapter 6 Future lines of work.

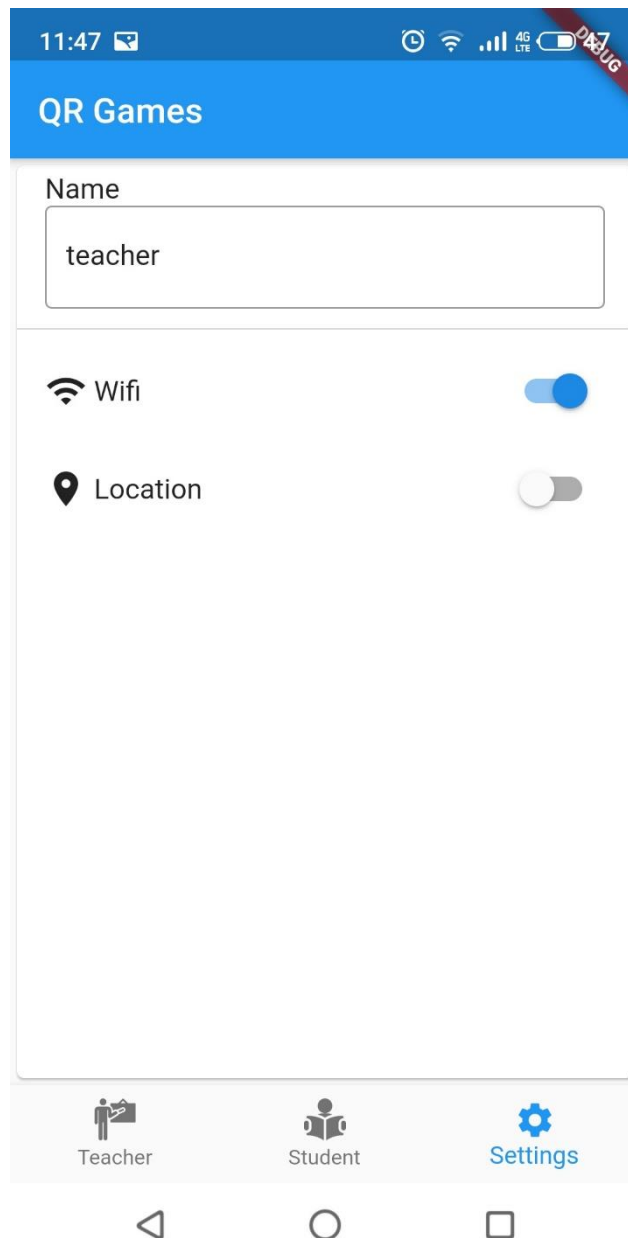


Figure 28 Settings view

3.4 Common module

In this module, all the classes affecting the devices memory will be detailed. As both the student and teacher are expected to make use of this functionalities, which will not be altered regardless of who is using them, the name common has been assigned to the module.

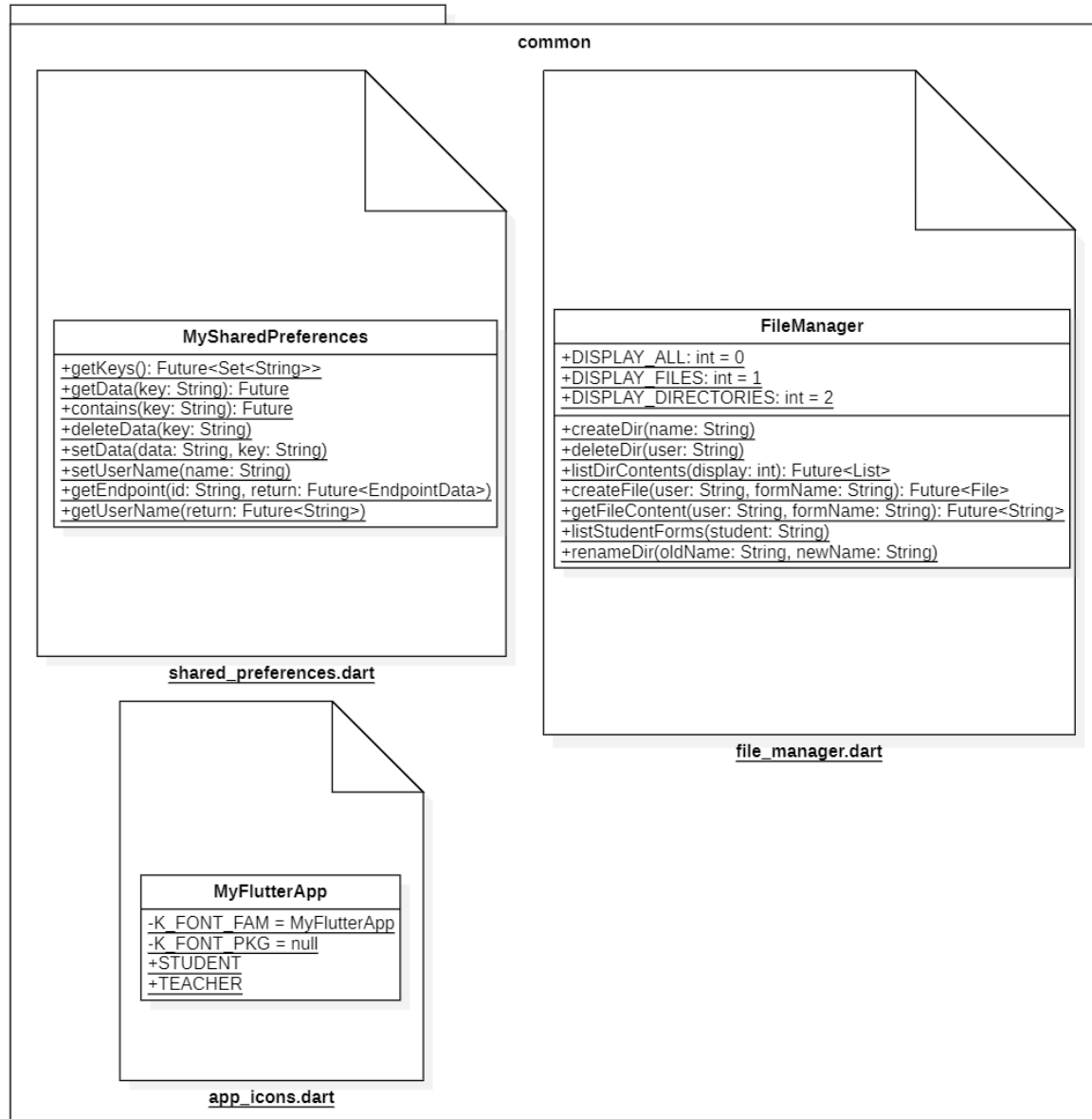


Figure 29 Common module UML diagram

3.4.1 File management

This class is centred around the usage of the `path_provider` plugin. This plugin offers the user navigation and alteration of the device's internal storage. Internal storage refers to the storage assigned to each specific app, which cannot be accessed by any other app or users. This storage space is deleted when its associated app is deleted[34].

Although the specific path may vary between android devices and versions (most notably since the introduction of separate user profiles in android 4.2)[35], its most common path is

/data/data/app.package.name/files, although this package makes use of the path /data/user/0, which is a symlink to /data/data[36].

In this project, a simple hierarchy of directories and files are used. When a new connection is received, a directory is created for that user. If it is a connection from a known device, his information is updated (student directory is renamed). After the initial connection, when receiving a response, the form received is stored in the corresponding folder, where the teacher may navigate to and review the answered form.

As of now, some functionalities have not been made available to the teacher, such as deleting folders or sorting by form rather than by user, but some of the necessary functions have been developed. Besides additional functionalities for the internal storage, this module is also focused to encompass all the necessary methods related to external storage, detailed in chapter 6 Future lines of work.

3.4.2 Shared preferences

While the file management allows for creation of directories and files, the shared_preferences plugin used in this module allows for the modification of one single file. While it may seem an underwhelming API to use, it is this way by design. It is true you can only modify a single file's contents asynchronously, the API provided offers simple methods to work with and it is designed to store a small collection data using a key-value format[37].

This allows for simple data storage and access, although the biggest flaw for shared preferences is the fact that there is no guarantee that writes will be persisted to disk after returning. While the easiness of use has made it a nice choice for this proof-of-concept application, upon release new data storage systems must be researched.

Regardless of its characteristics, in this project the shared_preferences plugin is used in several instances.

One instance is upon creation of a new form by the teacher. The form is modelled and transformed to a Json format. This Json format is the one stored in the shared preferences with the character “#” as a header to the key indicate it is a form. The key is the form's name.

Another instance is to store the user's name specified in the settings window. The user's name is stored under the key “username”.

Lastly, it is also used to store data regarding all the connected devices. This is where the device's information such as id, UUID, name and so on is stored. The key is the user's UUID.

The default path for the shared preferences xml file in your device is /data/data/YOUR_PACKAGE_NAME/shared_prefs/YOUR_PREFS_NAME.xml as displayed in Figure 30. Figure 30 displays an image from the teacher's shared preferences file, so that besides observing the shared preferences path and saved username, also the forms and the data from connected devices can be observed.

Must be noted that the data from connected devices is data from any connected device, not from currently connected devices. This is the data that is compared to the same user's data

upon reconnection to check if there have been any changes to the name, so that the previously defined file manager might apply any necessary modification to the user's directory.

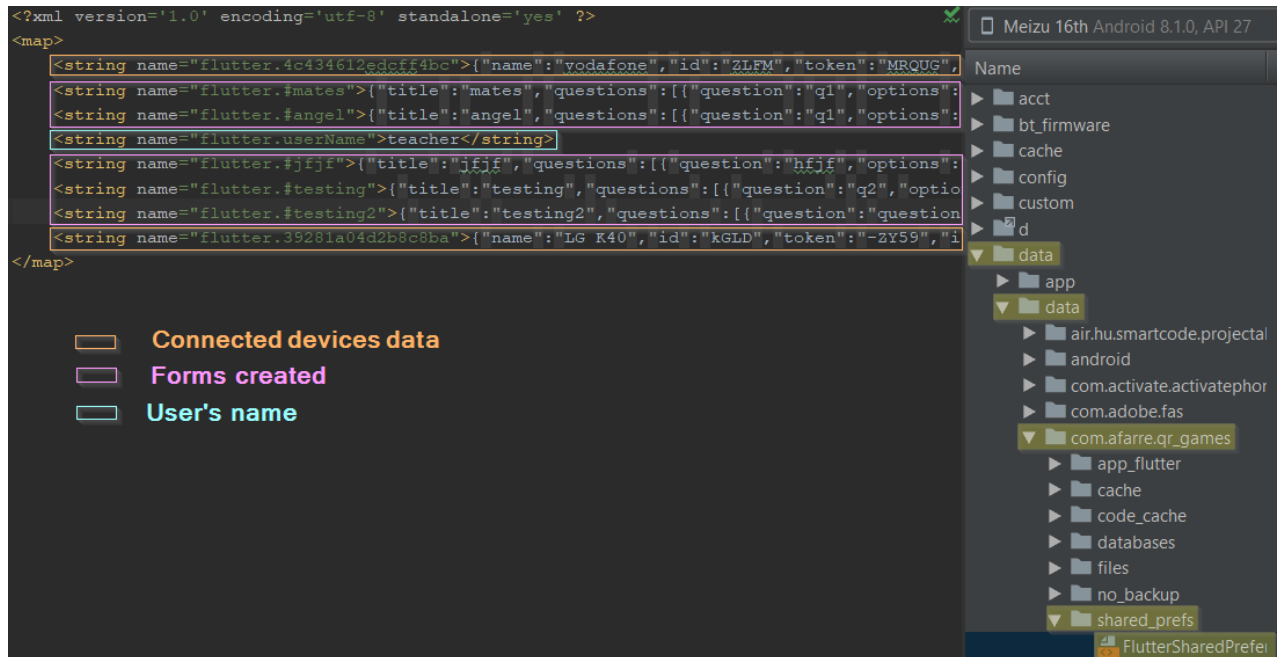


Figure 30 Shared preferences .xml file as displayed by Android Studio

3.4.3 App Icons

This dart file contains a simple class called MyFlutterApp. This self-generated class allows the usage of custom icons in the project. The portal used to generate these icons, FlutterIcon¹⁵, generates for the developer three files:

- A .dart file containing MyFlutterApp class. This is used to call the desired icons as such: *new Icon(MyFlutterApp.student)*.
- A TTF file. This file needs to be imported into the project via the pubspec.yaml file, under the font category.
- A json file. This file can be uploaded to the portal to load all the previously used icons and continue editing your set of icons. Serves no other purpose than to ease user experience.

The custom icons imported for this project are those of the teacher and student:

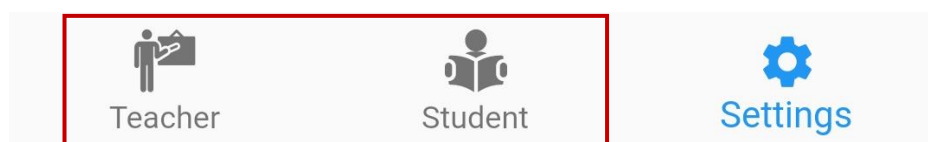


Figure 31 Custom Teacher and Student icons

¹⁵ <https://www.fluttericon.com/>

3.5 Model module

This module contains all the models used to implement the project. Each model, or set of models, are used in clearly distinctive sets of the project. For this reason, they have been distributed amongst three different .dart files, which are explained in more detailed in the three following chapters.

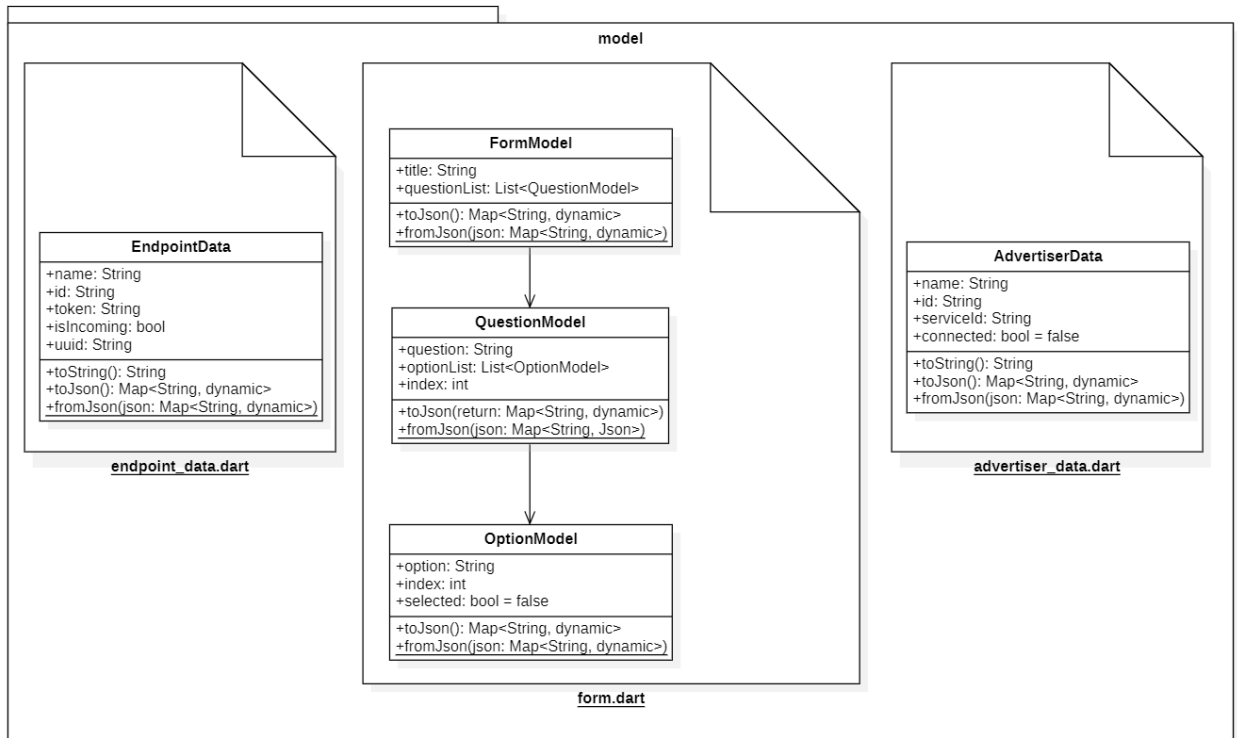


Figure 32 Model module UML diagram

3.5.1 Form

Regardless of whether the user sends or receives them, the forms shared amongst users must be stored. To store them however, they must first be modelled after a class.

For the forms, three classes have been created, rather than just one. The class FormModel, encloses all the forms. It has available a String field for the name and a list where all the questions that comprise the form, will be stored.

For each question within the form, a QuestionModel is created. It is made up of a String field for the name, an index to store what question it is and yet another list. This list however, is formed of OptionModels, to store all the information regarding all the options enclosed in a single question.

Finally, as mentioned, the OptionModel class is made up of a String field for the name, an index and also has a Boolean which indicates whether the option is selected or not. This last Boolean is utilised when the form is in the student's hands and when it is sent back to the teacher. Upon creation, its value is negligible.

All three classes have their corresponding methods fromJson and toJson, in order to cast forms to Json and vice versa. Displaying the form in its corresponding format (radioButtons, textFields...) is not modelled and is up to each process to cast the Json form as they desire. An example of a form json can be found in the following Figure 33.

```
{
  "title": "mates",
  "questions": [
    {
      "question": "q1",
      "options": [
        {
          "option": "o1.1",
          "index": 0,
          "selected": false
        },
        { ... },
        { ... },
        { ... }
      ],
      "index": 0
    },
    {
      "question": "q2",
      "options": [
        {
          "option": "o2.1",
          "index": 0,
          "selected": false
        },
        { ... }
      ],
      "index": 1
    }
  ]
}
```

Figure 33 Json of a form

Besides its use in displaying a form and data transmission from teacher to student, and backwards, this model is also utilised by both shared preferences and the file manager.

Regarding shared preferences, when a teacher creates a form, the newly created form is mapped onto this model and then stored into the shared preferences.

Regarding the file manager, when a student sends his response form to the teacher, upon receiving it, the teacher will model the student's answer using this model and will proceed to store the Json of the form in the corresponding student's folder in the file system.

3.5.2 Endpoint data

This class models an established connection on the teacher side. It utilises both data obtained upon initiating the connection between devices and from, once accepted the connection, a default data exchange.

The stored data from the connection initiation are the id, authentication token, name and a boolean indicating whether the connection is incoming or not. After the connection has been accepted between devices, by default the student sends his UUID to the teacher, which is also stored in the endpoint data. As explained in chapter 3.1.3 Endpoint list, this is because the id is only unique to the current connection. Upon connecting again, the device might have a different id, making identification of a previous connected device impossible. The UUID, unlike the connection id, is always the same for a given device.

In a similar fashion to the form model, the endpoint data model also has available the `fromJson` and `toJson` methods in order to cast any given instance to json, or vice versa. An example can be seen in the following figure

```
{
  "name": "teacher A",
  "id": "pRl7",
  "token": "AL_3U",
  "isIncoming": true,
  "uuid": "39281a04d2b8c8ba"
}
```

Figure 34 Json of an EndpointData

Endpoint data is only used on the teacher side and it is used to establish whether the connected device is a new device or one which had connected previously. In case it is a new device, the system will create a directory with the student's name where all his answered forms will be stored. In case it is an old device, the system will check whether the student has changed his/her name and will update the student's corresponding directory with the new name.

3.5.3 Advertiser data

This last model class serves a similar purpose to that of the endpoint data model, but for the student side. It is made up of a name, id, a service id, and a Boolean called `connected`. All of which are the parameters obtained when a new device is discovered, but not connected to, except the `connected` field, which is set by default to false.

This is important, as we have stated how nearby connections is unable to identify a previously connected device, hence the use of the UUID, but the UUID is sent by the student once the devices are connected. Upon discovery of a device, without establishing a connection, only the three aforementioned fields are available. This is the reason behind the fact that duplicates in the advertiser list may occur, as detailed in chapter 3.2.3 Building forms.

The `connected` Boolean simply serves the purpose to keep track of what device the student is currently connected to, so that a visual representation of the connection can be displayed in

the view. In this case, the fading colour surrounding the available devices, turns from black to green to indicate a live connection with that device.

The Advertiser data model class also contains the methods fromJson and toJson in order to cast data to Json format and viceversa, although as of now, the data modelled into the advertised model class is only utilised in run time, meaning neither shared preferences or file manager store any of it on the device. In any case, an example for a json file of an advertiser data can be seen in the following

```
{
  "name": "teacher A",
  "id": "4J7Q",
  "token": "com.pkmnapps.nearby_connections",
  "connected": false
}
```

Figure 35 Json of an AdvertiserData

4 Conclusions

In this project an application has been presented with the goal of increasing security in an educational field. This is achieved with the local communication of data, represented by a form, directly to the user's devices.

The analysis of several local transmitting data technologies has shown the existing limitations when it comes to connect multiple devices locally, without even taking into consideration the difficulties presented by cross-platform functionality.

As seen, this project has demonstrated how communicating several devices directly is possible although there is not, as of now, a free technology that allows multiple clients on multiple platforms to communicate. This has been an important limiting factor that has impacts in the final availability of the proposed solution.

While the main goal of the project has been achieved, personal goals have not.

On one side, local data transmission have been carried out successfully between multiple devices, both outgoing and incoming. The success of this proof of concept also means that security in the educational field can be easily increased without the heavy expenditure that comes alongside a full online protection.

It must be stated that Nearby connections has several security flaws of its own, as noted by researchers, "Google's Nearby Connections API is not only open to attack, but actively posing a threat to all Android applications using it"[38]. Although this analysis was performed in 2016 and Google was notified of it, it is unknown if version 2.0 of Nearby connections patched this security holes.

This, however, is of no major concern as the security strength of this technology comes from its local nature, not from an unbreachable communication channel between devices. By design, a user is safer using a local net than an external one, regardless of security measures.

On the other hand, the technology used, Google Nearby connections, has allowed for an implementation available only to the android devices, as iOS is not yet supported. However, Google nearby connections is constantly evolving and as said by its developers, multiplatform compatibility should be achieved soon.

As such, we can state that local communication is possible, even though the technology behind it (regardless of whether Google nearby connections, or another technology is being used) must mature further. Current state makes it useful for certain environments that depend on the presence of a router, or the number of users to connect to, but overall, it lacks flexibility.

5 Limitations

5.1 Technological limitations

As defined when analysing technologies in chapter 2.1.5.4 Nearby connections, there are two current limitations for this technology: inability to communicate with iOS devices and a limited number of connections.

The fact that currently nearby connections is **unable to communicate with iOS devices** is a major issue. Although Google is working on overcoming this barrier, there is not an estimated time of arrival for this feature, but it is a doable feature that will come with time (hopefully not too long).

It is true that several other technologies such as plain Bluetooth, BLE or Wi-Fi hotspots can connect to both iOS and Android devices, but this comes at the cost of the maximum number of connected devices.

Since Bluetooth, BLE and Wi-Fi hotspots have a **limited number of connections**, so does nearby connections which runs over these technologies.

As explained in chapter 2.1.5.4 Nearby connections, out of the three topologies nearby connections offers (P2P_STAR, P2P_CLUSTER, and P2P_POINT_TO_POINT), only P2P_STAR and P2P_POINT_TO_POINT make use of Bluetooth, BLE and Wi-Fi hotspots (P2P_CLUSTER uses BLE and Bluetooth only).

The topology chosen has been P2P_STAR, therefore the maximum number of connections is 10, given by the technology that can host the most devices: Wi-Fi hotspots.

Some may argue that the number should be greater, as for in the discovery phase, nearby connections discovers devices using Bluetooth and then upgrades them to the higher bandwidth Wi-Fi. If Wi-Fi hotspots has reached its maximum limit of 10 devices, then the remainder stay connected via Bluetooth, bringing the total up to 13-14 devices. While this statement is not entirely incorrect, it is also not entirely true.

Usually, Wi-Fi and Bluetooth share the same antenna on a device (Bluetooth running at 2.4GHz and Wi-Fi at 2.4/5GHz), meaning it has some scheduling going on. This means that when Bluetooth is using the antenna Wi-Fi will be put to sleep and vice versa.

Also, a limiting factor for the antenna is the fact that the device can only talk in one channel at the time and the antenna can only listen to one channel at the time. Meaning that it may miss some messages if the antenna is on the wrong channel at the wrong time. On top of it, the chip has limited RAM which may impact too on the number of devices[39].

As said, one could be forgiven for believing that the max number of devices is equal to the sum of all available technologies combined, as the hardware used (and other factors) limit it considerably.

In any case, to solve both problems at once, there are two choices:

- By using premium paid mesh services that some APIs offer, but this project is based on creating a proof-of-concept and as such, no funding has been provided, excluding any paid option available.
- Creating your own mesh. While it may be a cost-free solution, the time taken, and complexity of the solution should be considered.

While it will not solve the iOS connectivity issue, there is a solution to overcome the maximum number of connected devices in nearby connections. As William Harmon, a software engineer at Google, detailed[40]:

“To be able to connect all ~30 devices, I'd recommend forming a 'snake-like' connection. The head and tail of the device will scan and advertise at the same time (and devices that aren't connected to anyone are considered snakes of length 1). The heads and tails will keep connecting to each other (being sure not to connect to itself*), and you'll pretty quickly have a long chain of connections connecting everyone together. From there, you can forward messages down the chain to make sure everyone gets it.”

While William Harmon was unable to provide a code source for a snake-like connection, Dipayan Ray, an associate software engineer at Informatica, did provide an example for it¹⁶.

There is, however, a hidden feature that it is not stated on the documentation and that William Harmon kindly explained and that is that nearby connections will utilize a router if it is available for both devices.

So far, all limitations and solutions explained regarding the limitations of devices are in a presumed environment where a router is not present. In case of having an available router, then a series of actions are taken by nearby connections[39].

- During the discovery/advertise phase, nearby connections makes use of mDNS to scan for devices in the same Wi-Fi network, alongside a Bluetooth scan. The results are merged.
- After the initial connection, if the devices are connected via Bluetooth they share their SSID. If the shared SSID of the Wi-Fi network they are connected to is the same, and they are reachable, then it will attempt to use that connection instead of a hotspot, which is less disruptive.
- To use this connection, it will make use of TDLS (IEEE 802.11z standard) so that they can communicate directly with each other.

The usage of TDLS cannot be overstated. It not only allows for devices to communicate directly amongst them, but also:

- Data transmissions are secured with WPA-2 (unless the network is using an open, non-secured configuration, in which case the direct link is also set to open) or they are secured with a higher security protocol that both devices support.
- Allows the device to maintain connectivity to internet via the router.

¹⁶ <https://github.com/nSpider/NearbyConnectionMeshNetwork>

- Enables the devices to work at their highest level of shared capabilities.
- The devices can negotiate an alternative channel even if the network AP supports a lower bandwidth form of Wi-Fi.

5.2 Development limitations

Putting aside the technological limitations, there are some limitations that while they did not hamper the development of the application, could potentially be troublesome for the final user.

It has been mentioned that nearby connections implements a small protocol that shares all the mediums the devices support, so that it can upgrade to the best medium. Regardless of this, there is still a limitation on the number of data that can be sent over.

The official documentation for nearby communications states that when sending a byte payload, the maximum array number is limited to 32k[41]. The data sent in this project are forms, which are stored in the shared preferences. Shared preferences saves data under a key-value format, and the max value that can be saved is the same as the maximum length of a Java String, which is defined by the constant `Integer.MAX_VALUE` which equals $2^{31}-1$ [42].

Looking at these figures, we can state that the limit of the amount of data that can be sent is set by nearby connections itself, at 32k (32768). A reasonable number to offer to the user.

Another limiting factor is the device's OS. The minimum OS version should be Jelly Bean, or Android 4.3. Looking back at Figure 6, the number of devices that use Jelly Bean, or a higher version, is over 98.1% of the devices. With such a high figure, we can rule out that OS requirements might be an issue for the users.

Regarding the version of Google Play Services, the minimum version required is 11.0, which was released in 2017 (alongside connections 2.0). While there is no official data on Google Play Services distribution amongst Android devices, in 2014 Sundar Pichai made an official statement in the 7th annual Google I/O developer conference, indicating that 93% of users are on the latest version[43], which currently is 20.47.13. So, although not having up to date data on Google Play Services, it is an educated guess that it will not be a limiting factor, and even if that was the case, the user needs only to update this service via the App Store application.

6 Future lines of work

While the goals of the project have been achieved, there are many improvements and new features that can be added to the project. Some of the next steps, in order of importance, are as follows.

Firstly, and most importantly, known bugs must be fixed. These bugs are:

- Deleting a concrete question in a form.
- Inability to connect to a device once it has been discovered but is not advertising any more.
- Inability to send a form once both devices have established a connection, but the student closes and opens the application again.
- Duplication of devices in the student's discovery list as there is no unique ID to identify a teacher that advertises twice under different IDs.

Upon fixing all issues that might affect the correct workflow of the application, the next step is to beautify the project. This means embellishing the views and making the overall navigation through the project more user friendly or adding alert dialogs when certain critical action is being performed or when the user must be informed of something.

Since this is an educational project, public might include pupils from a young age as well as elder staff. Making the application as accessible and as easy to use for all the targeted profiles is a must.

Once the project has reached this state, it can be considered ready for release as it has a functioning feature and is user friendly.

After the release milestone, the next steps include:

Fixing problems that might arise as some specific device might have incompatibility issues or similar. Not only can issues arise from this application's development, but also in dependencies such as that of plugins or from nearby connections itself. As an example, issues have been reported when using nearby connections in android 10. This can be boiled down to one word, maintenance.

As a last future line of work, once the application has reached release status, a series of upgrades can be implemented to offer new functionalities. Some of these functionalities might include:

- Sort student forms by form, not by student, on the teacher side.
- Offer multiple elements to build a form, other than radio buttons, such as text forms, a space for the user to draw, upload documents alongside the form...
- Include the use of images in the advertiser/discover lists.
- Add a tracking system for attendance.
- Reworking the settings view to include changing the path of the student's directories, enable dark mode, select user's picture (presuming the previous point has been implemented), changing the language...

- Allow the student to store received forms and access them later to fill them up off-site, and then send them to the teacher later.
- Allow the teacher to automatically accept incoming connections.
- Allow the teacher to automatically grade the student's form.
- Implement iOS compatibility once it is supported by nearby connections and make any necessary changes to the project to avoid issues amongst devices.
- Implement MVVM architecture[44].

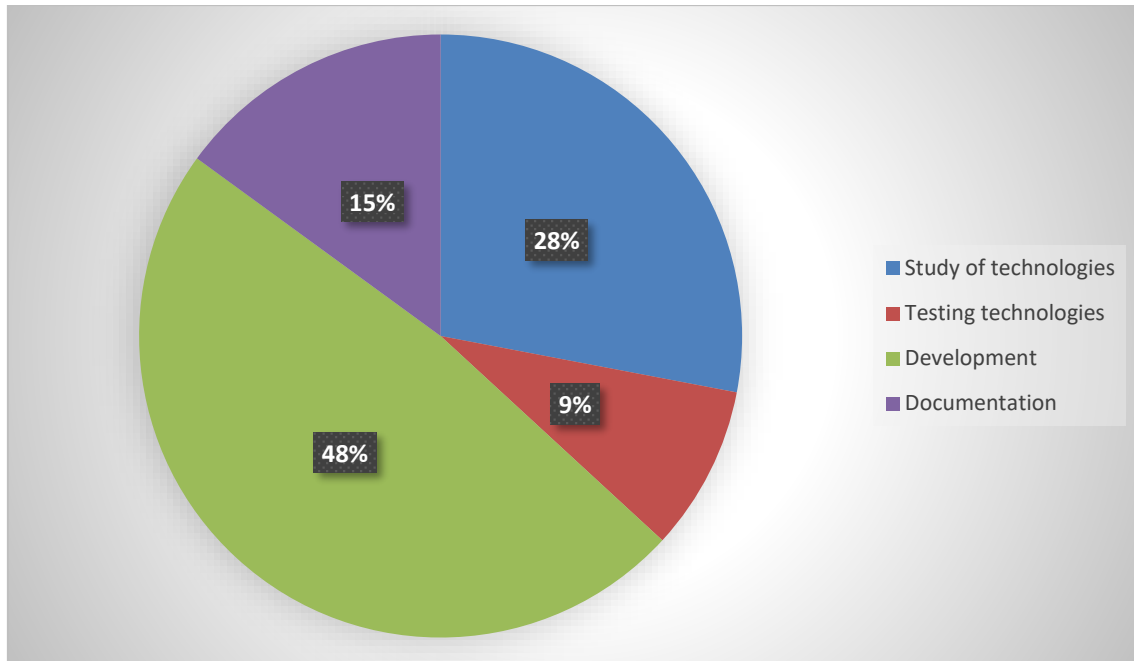
This is all regarding what features and fixes are strongly recommended for the future of this project. Testing, however, must not be overlooked. This application needs a thorough testing process in order to determine two main points.

Firstly, several iterations to study user navigation. A well-done untested design can still have many flaws that the user might encounter that cannot be apparent to the developer and need to be detected and fixed via an extensive iterative process.

Secondly, to determine the limitations of the project. Some limitations have been established in the chapter 5 Limitations, but some such as the max number of connected devices and max distance, are theoretical and need to be tested in a real environment.

7 Time estimation

The following graph showcases an estimation regarding the time invested into this TFG. The total estimated cost in time is of 768 hours.



From greater to less, a great part of the project was the development itself. In hindsight, developing such an application does not require such a time investment for a seasoned developer, but the limited knowledge of flutter prior to this project made the cost in time rise.

Following the development time, the study of technologies undertaken before even creating the flutter project holds the second place. As the requirements have not specified, looking for the appropriate technology was paramount for a successful development of the project.

Thirdly, the documentation of the project. While maybe not having a significant cost in time such as the two previously mentioned steps of the project, it is nonetheless an essential part of the project.

Finally, and closely related to the technology's investigation, testing was done both using several of the possible technologies and within one technology, the testing of its associated plugins to select the one that suited the project's needs and worked best.

8 References

- [1] HoloniQ, '§16.1B of Global EdTech Venture Capital in 2020 – HoloniQ', 2021. <https://www.holoniq.com/notes/16.1b-of-global-edtech-venture-capital-in-2020/> (accessed Jan. 09, 2021).
- [2] Bitdefender and Dan-Mihai Iorgulescu-Stavri, 'Threat Landscape Report 2020', 2020. Accessed: Jan. 09, 2021. [Online]. Available: www.bitdefender.com.
- [3] Impact, '10 Cybersecurity in Education Stats You Should Know for 2020', 2020. <https://www.impactmybiz.com/blog/cybersecurity-in-education-stats-2020/> (accessed Jan. 09, 2021).
- [4] T. Riley, 'The Cybersecurity 202: Global losses from cybercrime skyrocketed to nearly \$1 trillion in 2020, new report finds - The Washington Post', 2020. <https://www.washingtonpost.com/politics/2020/12/07/cybersecurity-202-global-losses-cybercrime-skyrocketed-nearly-1-trillion-2020/> (accessed Jan. 09, 2021).
- [5] B. Kelly, L. Ryan M., and D. C. Paolo, 'State of Cybersecurity Report 2020 - Accenture Security', 2020. Accessed: Jan. 09, 2021. [Online]. Available: https://twitter.com/Paolo_DalCin.
- [6] D. Amo, D., Torres, R., Canaleta, X., Herrero-Martín, J., Rodríguez, C., & Fonseca, 'Seven principles to foster privacy and security in educational tools: Local Educational Data Analytics. TEEM'20', 2020. <https://www.youtube.com/watch?v=-62RIFgmiIJ&feature=youtu.be> (accessed Jan. 09, 2021).
- [7] D. Seweryn, 'A Curious Relationship: Android BLE and Location | Polidea', 2019. <https://www.polidea.com/blog/a-curious-relationship-android-ble-and-location/> (accessed Oct. 21, 2020).
- [8] M. Cunche *et al.*, 'On using Bluetooth-Low-Energy for contact tracing On using Bluetooth-Low-Energy for contact tracing. [Research Report] Inria Grenoble Rhône-Alpes; INSA de Lyon On using Bluetooth-Low-Energy for contact tracing', 2020. Accessed: Dec. 22, 2020. [Online]. Available: <https://hal.inria.fr/hal-02878346v3>.
- [9] Brendan, 'List of Android Wi-Fi Direct Documentation and Resources', 2016. <https://groups.google.com/g/wi-fi-direct/c/uWpuOzHY6y0> (accessed Nov. 11, 2020).
- [10] Thali project, 'Android Wireless Issues'. <http://thaliproject.org/androidWirelessIssues/> (accessed Dec. 22, 2020).
- [11] B. Nelson, 'The Challenges of Soft AP: What Goes Wrong and Why', 2017. <https://blog.cirrent.com/challenges-soft-ap> (accessed Nov. 09, 2020).
- [12] B. Ray, 'Bluetooth Vs. Bluetooth Low Energy (BLE): What's The Difference?' <https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy> (accessed Oct. 17, 2020).
- [13] R. Castro, 'Bluetooth Classic Vs. Low Energy Vs. Smart Vs. Smart Ready ¿Cuál es cuál?', 2020. <https://www.wikiversus.com/informatica/bluetooth-classic-vs-bluetooth-low-energy/> (accessed Dec. 22, 2020).
- [14] SIG, 'Traditional Profile Specifications | Bluetooth® Technology Website', 2021. <https://www.bluetooth.com/specifications/profiles-overview/> (accessed Dec. 22,

2020).

- [15] 'Advantages and Disadvantages of Infrared sensor', 2012. <https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Infrared-Sensor.html> (accessed Dec. 22, 2020).
- [16] B. Armstrong, 'quiet/org.quietmodem.Quiet: Quiet for Android - TCP over sound', 2020. <https://github.com/quiet/org.quietmodem.Quiet> (accessed Dec. 22, 2020).
- [17] Google Developers, 'Nearby | Google Developers'. <https://developers.google.com/nearby> (accessed Dec. 26, 2020).
- [18] M. Rahman, 'Google's Nearby Share, its file-sharing AirDrop clone for Android, rolls out', 2020. <https://www.xda-developers.com/google-nearby-share-file-sharing-airdrop-clone-android-rolling-out/> (accessed Oct. 25, 2020).
- [19] D. Bohn, 'Android's "Nearby Share" file sharing feature is finally launching - The Verge', 2020. <https://www.theverge.com/2020/8/4/21353020/android-nearby-share-file-sharing-feature-launch-airdrop> (accessed Oct. 25, 2020).
- [20] Microsoft, 'Android Nearby sharing API reference - Project Rome | Microsoft Docs', 2019. <https://docs.microsoft.com/en-us/windows/project-rome/nearby-sharing/api-reference-for-android> (accessed Dec. 26, 2020).
- [21] Google Developers, 'Nearby Notifications Overview | Google Developers', 2015. <https://developers.google.com/nearby/notifications/overview> (accessed Dec. 28, 2020).
- [22] J. Adkins, B. Ghena, and P. Dutta, 'Freeloader's Guide Through the Google Galaxy', 2019, doi: 10.1145/3301293.3302376.
- [23] R. Nayak M, 'Android Developers Blog: Discontinuing support for Android Nearby Notifications', 2018. <https://android-developers.googleblog.com/2018/10/discontinuing-support-for-android.html> (accessed Dec. 28, 2020).
- [24] 'Overview | Nearby Messages API | Google Developers'. <https://developers.google.com/nearby/messages/overview> (accessed Dec. 28, 2020).
- [25] Google Developers, 'Nearby Messages (100 days of Google Dev) - YouTube', Jul. 17, 2015. <https://www.youtube.com/watch?v=hultDpBS22s&feature=youtu.be> (accessed Dec. 28, 2020).
- [26] M. C. Shepherd, 'Manatee: Using Google Nearby Messages to Build a Cross-Platform, Proximity-Based Mobile Client for Cards Against Humanity-Style Party Games', 2019. <https://nrs.harvard.edu/URN-3:HUL.INSTREPOS:37364571> (accessed Nov. 06, 2020).
- [27] Google Developers, 'Android Developers Blog: Announcing Nearby Connections 2.0: fully offline, high bandwidth peer to peer device communication', 2017. <https://android-developers.googleblog.com/2017/07/announcing-nearby-connections-20-fully.html> (accessed Nov. 08, 2020).
- [28] 'Overview | Nearby Connections API | Google Developers'. <https://developers.google.com/nearby/connections/overview?authuser=1> (accessed Nov. 21, 2020).

- [29] S. Bansal, 'Nearby Connection API. Nearby connections is a peer-to-peer... | ProAndroidDev'. <https://proandroiddev.com/nearby-connection-api-b235529e6643> (accessed Nov. 12, 2020).
- [30] W. Harmon, 'Google Nearby Connections 2.0 capabilities - Stack Overflow', 2018. <https://stackoverflow.com/a/51997757/7060082> (accessed Jan. 05, 2021).
- [31] E. Proj, 'Android Nearby, Connections vs. Messages| Mobile App Development Publication | Medium', 2016. <https://medium.com/mobile-app-development-publication/android-nearby-connections-vs-messages-a2bdf6a59ff3> (accessed Dec. 28, 2020).
- [32] Google Developers, 'Google Play | Android Developers', 2020. <https://developer.android.com/distribute/best-practices/engage/nearby-interactions#best-practices> (accessed Jan. 05, 2021).
- [33] R. Rastel, 'Dart: What are mixins?. It's a kind of magic 🪄 | Flutter Community | Medium', 2018. <https://medium.com/flutter-community/dart-what-are-mixins-3a72344011f3> (accessed Jan. 12, 2021).
- [34] Android developers, 'Data and file storage overview', 2020. <https://developer.android.com/training/data-storage#filesInternal> (accessed Jan. 04, 2021).
- [35] D. Hulme, 'Internal storage - What kind of data is stored in /data/user directory? - Android Enthusiasts Stack Exchange', 2013. <https://android.stackexchange.com/a/48397> (accessed Jan. 04, 2021).
- [36] CommonsWare, 'The Storage Situation: Internal Storage', 2017. <https://commonsware.com/blog/2017/11/13/storage-situation-internal-storage.html> (accessed Jan. 04, 2021).
- [37] Android developers, 'Save key-value data | Android Developers', 2020. <https://developer.android.com/training/data-storage/shared-preferences> (accessed Jan. 05, 2021).
- [38] D. Antonioli, N. O. Tippenhauer, and K. Rasmussen, 'Nearby Threats: Reversing, Analyzing, and Attacking Google's "Nearby Connections" on Android', 2016, doi: 10.14722/ndss.2016.23xxx.
- [39] W. Harmon, 'Nearby Connections max connected devices - clarification - Stack Overflow', 2021. <https://stackoverflow.com/a/65588012/7060082> (accessed Jan. 06, 2021).
- [40] W. Harmon, 'Be able to send Messages/Bytes Simultaneous to multiple devices using Nearby Connections - Stack Overflow', 2018. <https://stackoverflow.com/a/52785805/7060082> (accessed Jan. 05, 2021).
- [41] Google Developers, 'ConnectionsClient | Google APIs for Android | Google Developers', 2020. https://developers.google.com/android/reference/com/google/android/gms/nearby/connection/ConnectionsClient?authuser=1#MAX_BYTES_DATA_SIZE (accessed Dec. 07, 2020).
- [42] J. Yingst, 'Maximum length of String (Beginning Java forum at Coderanch)', 2004.

<https://coderanch.com/t/393042/java/maximum-length-String> (accessed Dec. 05, 2020).

- [43] S. Pichai, 'Google I/O 2014 - Keynote - YouTube', 2014. <https://www.youtube.com/watch?v=wtLJPvx7-ys&feature=youtu.be&t=45m35s> (accessed Jan. 05, 2021).
- [44] S. Sharma, 'An Introduction To MVVM Architecture In Flutter - Appventurez', 2020. <https://www.appventurez.com/blog/introduction-mvvm-architecture-flutter/> (accessed Dec. 04, 2020).