FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Real-Time Assistance for Visually Impaired Individuals
Student:	Ernesto Iván Ochoa Hidalgo
Supervisor:	Ing. Erik Derner, Ph.D.
Study program:	Informatics
Branch / specialization:	Software Engineering 2021
Department:	Department of Software Engineering
Validity:	until the end of summer semester 2023/2024

Instructions

Mobile applications can assist visually impaired people in performing a wide range of daily tasks, including navigation in unknown environments. One category of these applications is based on establishing a connection with a fully sighted user to provide assistance. While a few applications offering real-time support by transferring information from the visually impaired user's smartphone to a fully sighted user are available, they have certain drawbacks that complicate their use for the target group. Existing applications are missing useful features such as high-resolution static image sharing, they do not allow for efficient control, or they are marketed at a high price.

This thesis aims to survey existing applications for sharing audiovisual and location information from the visually impaired user's smartphone with a fully sighted user, identify the drawbacks of such applications, and propose a solution that overcomes these limitations. In particular, the thesis is expected to address the following objectives:

1. Survey existing mobile applications offering the assistance of fully sighted users to visually impaired users and analyze their advantages and disadvantages.
2. Identify the most critical missing features among the available solutions.
3. Design, implement, and evaluate a prototype of an application offering the assistance of fully sighted users to visually impaired users, addressing the identified drawbacks of the existing applications.

Electronically approved by Ing. Michal Valenta, Ph.D. on 11 December 2022 in Prague.

Master's thesis

REAL-TIME ASSISTANCE FOR VISUALLY IMPAIRED INDIVIDUALS

Bc. Ernesto Iván Ochoa Hidalgo

Faculty of Information Technology
Department of Software Engineering
Supervisor: Ing. Erik Derner, Ph.D.
May 4, 2023

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis: Ochoa Hidalgo Ernesto Iván. *Real-Time Assistance for Visually Impaired Individuals.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Contents

List of Figures

List of Tables

List of code listings

I would like express my deepest gratitude to my thesis supervisor, Ing. Erik Derner, Ph.D., whose guidance and support have been invaluable throught my academic life and the development of this project. I would also like to thank the faculty and staff of the Czech Technical University in Prague who provided me with an exceptional learning environment and access to endless opportunities for personal and professional growth. Special thanks go to the International Student Club whose members have aided me throught my studies and have provided me with so many unforgettable memories. I would like to extend my appreciation to my friends and family, in particular my parents Ernesto and Teresa whose unwavering support has allowed me to embark on the study program that culminates with this work. I would also like to acknowledge the time and support provided by the admirable members of Czech Blind United which was an invaluable source for guidance and feedback during the development of this thesis.
A huge thank you to everyone who contributed to this project in one way or another, your support and encouragement have made this achievement possible.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 4, 2023 .....

Abstract

This thesis presents the results of research on the tools and accessibility features currently available for visually impaired people. We have identified key points for improving existing solutions and produced a web application as a result of four prototype iterations. This project was developed closely with the input of the Czech Blind United association members. The application has been tested on various platforms and refined based on the feedback from visually impaired users. The result of this work is a publicly available application accessible via a web browser whose features were requested by visually impaired users. This thesis describes the technical structure of this application and the technologies used in its implementation. By means of this work, we hope to promote awareness of the experiences and needs of visually impaired people and showcase that their lives can be improved through the use of technology.

Keywords Visually impaired users, video calls, real-time assistance, Flutter, Czech Blind United.

Abstrakt

Tato práce prezentuje výsledky výzkumu nástrojů a funkcí dostupných pro zrakově postižené osoby. Identifikovali jsme klíčové body pro zlepšení existujících řešení a vytvořili webovou aplikaci, která je výsledkem čtyř iterací prototypů. Tento projekt byl vyvinut ve spolupráci se členy Sjednocené organizace nevidomých a slabozrakých ČR (SONS). Aplikace byla testována na různých platformách a upravována na základě zpětné vazby od zrakově postižených uživatelů. Výsledkem této práce je veřejně dostupná aplikace, která je přístupná prostřednictvím webového prohlížeče a jejíž funkce byly požadovány zrakově postiženými uživateli. Tato diplomová práce popisuje technickou strukturu této aplikace a technologie použité při její implementaci. Prostřednictvím této práce bychom rádi přispěli ke zvýšení povědomí o zkušenostech a potřebách zrakově postižených osob a ukázali, že jejich život mohou usnadnit technologie.

Klíčová slova Zrakově postižení uživatelé, videohovory, asistence v reálném čase, Flutter, Sjednocená organizace nevidomých a slabozrakých ČR.

List of Abbreviations

DOM	Document object model
PWA	Progressive web application
SDK	Software development kit
SONS	Sjednocená organizace nevidomých a slabozrakých (Czech Blind United)
TURN	Traversal Using Relays around NAT
VPN	Virtual Private Network
W3C	World Wide Web Consortium

Chapter 1
Introduction

What is the situation for visually impaired people in the Czech Republic? How is sight affected by different conditions and what are their consequences? We will explore these questions and explain our work with the Czech Blind United association.

The number of visually impaired people in the Czech Republic has a positive growth tendency. In this chapter, we will explore how different types of sight conditions can have different consequences for the person that has them and how their treatments can range from simple non-invasive solutions such as wearing lenses to the requirement of corrective surgery. We will also explain the invaluable collaboration with the members of SONS as well as key insights collected through discussion of their experiences, needs and the limitations they have encountered while using the applications which are available today.

1.1 Visual Impairment in the Czech Republic, Figures, Types and Observations

Visually impaired people in the Czech Republic were at an estimated number of 1.8 million in 2020 of which 28,000 were classified as completely blind [1]. The crude rate of prevalence, obtained by dividing the number of visually impaired individuals by the population total, of people with vision loss in the Czech Republic has risen by 4.4 % since 1990, see Figure 1.1.

Not all visual impairments are the same, the treatments and consequences of each type of visual deficiency are varied. Refractive errors are a visual condition in which it is not possible to properly focus the eyes producing a blurry image, which can be caused when the shape of the eye prevents light from focusing correctly on the retina [2]. This is the largest cause of vision loss being responsible for an estimated 161 million people living with long distance vision impairment and 510 million people with near vision impairment worldwide [3]. In most cases, the refractive error can be treated by wearing glasses, contact lenses or by performing refractive surgery.

Another type of visual impairment is cataracts which cause a cloudy area in the lens of the eye which causes blurry vision, faded colors, difficulty seeing on low light conditions and making lamps or other sources of light to appear extremely bright [4]. Cataracts can only be fully treated with the use of surgery.

Sight is affected in different ways depending on the type of condition that each person

has, for this reason there is no single technological solution that can cover all the different needs of visually impaired users. Some users with a sight impairment can read text on a screen without making any adjustments while others may need to increase the font size or contrast of the colors displayed, others may make use of additional tools for aid, some of them may be particularly oriented towards use by people with sight conditions such as magnifying glasses or refreshable braille displays while others may be of more general use, such as portable Bluetooth keyboards, to properly operate their devices.

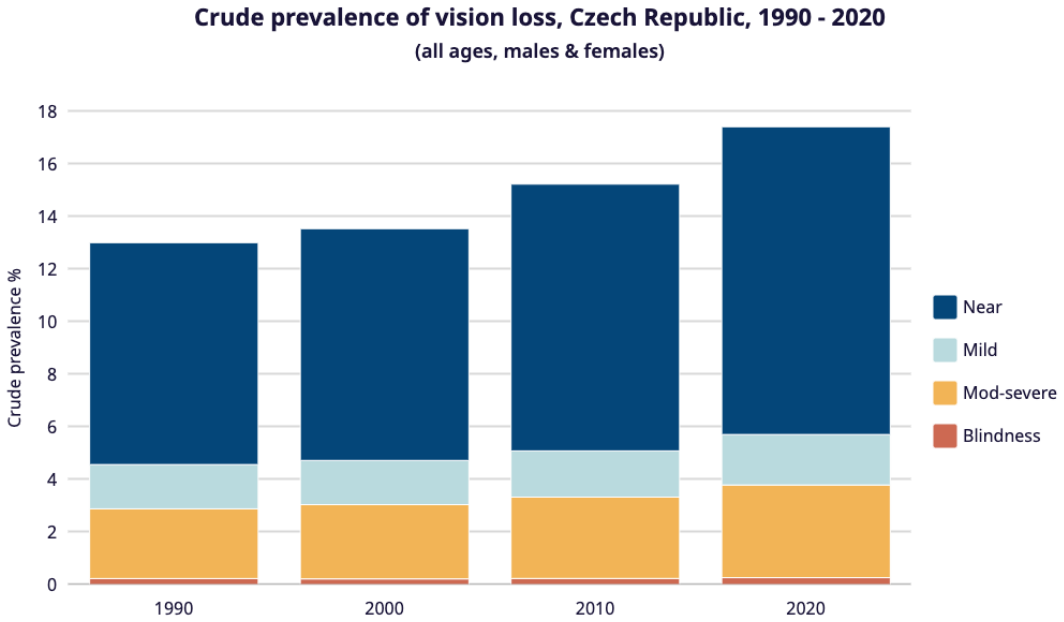

Figure 1.1 Crude prevalence of vision loss, Czech Republic, 1990–2020. Sourced from data from the VLEG/GBD 2020 model, accessed via the IAPB Vision Atlas [1].

The challenges that visually impaired users face in order to remain as active, creative and productive as their sighted peers are numerous and the tools that have been developed for this specific user group are currently not sufficient due to the limited number of accessible solutions both in financial and usability terms. In this project, I have combined years of software development experience with my drive to provide solutions that are both usable in the real world and that empower the people that use them.

1.2 Meetings at SONS

Established in 1996, SONS (*Sjednocená organizace nevidomých a slabozrakých ČR* in Czech; Czech Blind United) is a nationwide registered association in the Czech Republic with the objective of assisting individuals who are partially sighted or fully blind. The organization offers a wide range of services, including orientation, support, advice, training, employment assistance, club activities, and guide dog training [5].

To ensure that this project is successful, we worked closely with them and incorporated their feedback throughout the development process. We participated in periodic meetings with SONS members during which they provided us with valuable insights into their needs,

habits and limitations of existing solutions. In this section, we summarize some of these observations.

- Applications that read text using the camera are constantly used to accomplish daily tasks such as reading letters and checking the expiry date of grocery products.
- Existing applications sometimes charge prices that do not justify their continued use. While some of the offered functionalities are extremely useful, the frequency at which they are needed is too low to justify a subscription for a prolonged amount of time.
- Use of both applications designed for local transport systems such as Můj vlak and applications designed for worldwide use such as Google Maps is common. Accessibility features in some mainstream applications are sufficient to allow their use with relatively few inconveniences.
- It is not uncommon for applications to have significant drawbacks, for example there is an application used for telling the state of a traffic light that sometimes will register a traffic light located further back or in another crossing which could easily result in an accident when crossing the street relying purely on the application.
- Some sophisticated mainstream applications are accessible but the user experience is quite poor, frequently a simple action takes too long due to the integrated voice reader going over too many elements in the screen at once or requiring too many interactions to progress.
- Navigation aids are plentiful, but the more useful ones have been discontinued or are quickly becoming obsolete due to the lack of support for blind users or the reliance on outdated information about transport lines. Some applications offer textual descriptions of the route the user wishes to follow but these are not sufficient to navigate cities like Prague where the layout of the streets very often does not follow a grid-like pattern.
- Peripherals designed to work with the city infrastructure such as remote control devices that alert the user when a specific tram or bus is arriving often do not work due to a technical malfunction or the drivers themselves turning off the receiver for such devices.
- Certain technical limitations are usually not taken into account during the user experience, for example mobile data can be consumed very quickly if streaming video to a fully sighted assistant without a way to regulate the quality of the video or the GPS location can often be inaccurate in urban environments.

During our meetings, we discussed the drawbacks of current applications, desirable features and characteristics, and gathered feedback from users with different levels of visual impairments and habits. Their different perspectives and experiences based on their individual preferences when selecting and making use of accessibility solutions have provided a broad pool of anecdotal experiences that allowed us to think about how a solution should be designed, not only for the fully blind but also for the partially sighted, the people assisting them and for the developers who are considering making an application with accessibility as the primary goal.

Chapter 2

Review of Existing Applications

What software solutions are currently available for people with sight problems? How can these be categorized and what are their advantages and disadvantages?

In this chapter, we have researched technological solutions that help improve the experience of the visually impaired user when using an application or surfing a web page. Our research covers both the accessibility features built into devices and third-party applications explicitly designed for visually impaired users. In addition, we incorporate feedback from members of SONS who shared their personal experiences and observations.

2.1 Native Mobile Accessibility Options

Visually impaired users take advantage of features that are already present in mobile devices in order to navigate applications and perform daily tasks on their devices. While these features are sufficient for most popular applications, their effectiveness is constrained by the level of support implemented by individual applications. This frequently results in a user experience that is accessible but not efficient or intuitive.

During our meetings with SONS we observed how visually impaired users interact with these features. We have condensed some of the observations gathered during this process.

- Two modes are used for navigating, the 'manual' mode which cycles through interactable elements on the screen sequentially and the 'explore' mode which works by dragging a finger through the screen while the phone describes what is currently being selected. Both modes can be active at the same time. This can have, as a consequence, added difficulty when navigating through items displayed on the screen due to the fact that one wrong tap can make the navigation selector skip multiple elements and potentially disrupt the experience of the user.

- There exists an option that will list all of the available controls on the screen and present them in order. The user can then select the element they wish to interact with from this list. Unfortunately, it is not uncommon to encounter elements that do not possess a description or are completely inaccessible by this option.

- Consistent layouts are helpful for users when learning new applications since they can quickly locate elements such as drop-down menus and text boxes if they are located in similar positions across multiple applications.
- Extra hardware such as Bluetooth keyboards are sometimes used in tandem with these accessibility features.
- Typing using the standard on-screen keyboard is very slow due to the voice-over narrator not pronouncing the highlighted key clearly and having to wait for a second narration of a word that starts with the selected character. The user experience with this feature is largely dependent on the quality and type of the voice selected. Some of them enunciate certain letters in a similar way, such as with the letters 'V' and 'B', which results in the user having to spend time learning to discern the difference between the two or wait for the narrator to use the letter in an example before continuing.
- Changing the font size is inconsistent, some websites and applications are unusable due to the layout being changed and elements being hidden when the font size is expanded beyond a certain percentage. Some applications handle text elements in such a way that they behave differently from the device settings resulting in them being incompatible with this feature. Ideally, users would like to decide on the font size increment on an application-to-application basis instead of having a global setting.
- When navigating through the elements one by one, there is a possibility that the navigator will enter a cycle where the same elements are described circularly. For applications with a lot of interactable elements, a simplification of the elements that the navigator will describe would be helpful. In this way, the user can quickly navigate through the most important controls while skipping over elements that are not frequently needed.

2.1.1 Android Native Accessibility Options

Android devices come with a set of accessibility options ready to be activated from the moment the user begins to interact with the device. The following are the features offered by this operating system [6]:

- **Talkback:** Provides an audio description of the element that is currently highlighted or is being touched, describes the actions the user is performing and alerts about upcoming notifications.
- **Braille keyboard:** Enables a braille keyboard on the screen that allows the user to input text using a 6-dot braille interface. An example of such a keyboard can be seen in Figure 2.1.
- **Display and font:** The size of items on the screen, the size of the window used for displaying items on the screen and the font size can be changed.
- **Magnification:** Specific sections of the screen can be temporarily enlarged.
- **Select to speak:** The user can select specific items on the screen or point the camera at objects to hear them read or described aloud. Supports reading items in an application while the user switches to a different application, after which Select to speak will continue reading items in the background.

- **Lookout:** Uses the camera and sensors of the device to get details about the environment around the user, can be used to read printed text aloud, provide a basic description of the environment, scan food labels and recognize packaged food products, scan printed or handwritten documents and detect the value of bills. This feature must be downloaded from Google Play and is only supported on devices running Android 6.0 or higher.

- **Voice access:** Allows the device to be controlled with voice commands. The commands can be used to open applications, navigate them and edit text.

- **Switch access:** Interaction with the device can be carried out with external devices such as switches or keyboards instead of the touchscreen.

- **Action blocks:** Assigns routine actions to buttons on the device's home screen.

- **Time to take action:** Changes the time that messages requiring an action from the user are displayed. Affects items such as pop ups or push notifications that are on screen for a few seconds at a time. This option is not supported by all applications and is only available on devices running Android 10.0 or later.

- **Captions:** Speech on the device can be translated into captions in real-time, allows speech and sound to be captured and displayed as text on the screen, and enables text as a communication channel on phone calls. This feature is intended to be used by deaf users.

- **Audio:** Adds support for wired or Bluetooth headphones to filter, augment and amplify sounds in the environment or the device.

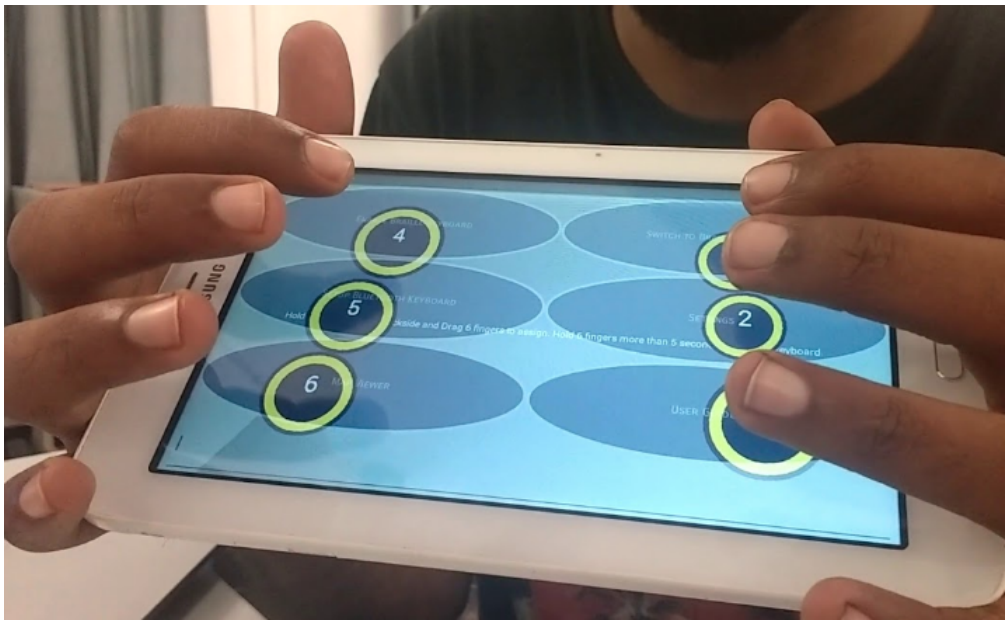

Figure 2.1 An example of an on-screen Braille keyboard, obtained from [7].

2.1.2 iOS Native Accessibility Options

Devices that run the iOS operating system such as iPhones or iPads have the following accessibility features available [8]:

- **VoiceOver:** Screen reader that describes the task being currently performed on the device, enables gestures for navigation. The speaking rate and pitch of the voice can be adjusted. The language of the voice reader can also be configured.
- **Typing feedback:** When typing on the device, letters and words will be spoken, auto-corrections and capitalizations are also announced. The user can touch and hold on a word to hear text predictions.
- **Audio descriptions:** When watching movies on the device, scenes can be described using this feature. Can appear by default.
- **Magnifier:** Allows the device to be used as a magnifying glass so the user can zoom in on objects near them. Useful for reading small text, applying filters to increase visibility and saving magnified screenshots to the device.
- **Display and text size:** Color inversion is provided, the layout of content, font size, color, intensity and tint can be changed. Certain devices allow for frame rate customization.
- **Zoom:** A permanent zoom can be configured, the entire screen can be zoomed or certain parts can be highlighted with a resizable lens. Compatible with VoiceOver.
- **Reduce motion:** Allows motion effects to be disabled on the device.

2.2 Remote Assistance Applications

In order to receive real-time assistance, a variety of applications are used. Some common ones are mainstream video and audio call applications that are not primarily targeted towards users with visual impairments, such as WhatsApp, Skype and Messenger. There are also applications that are designed specifically towards this type of users. One of the most popular examples of such applications is Be My Eyes, a mobile application that connects blind or partially sighted users with fully sighted users through a live video call. This application is free to use and allows users to choose their languages of preference. In order to provide assistance, the application does not require a formal registration and there is no option to see calls that are on hold. The phone will automatically start ringing when there is a user requesting assistance. When an user registers as a visually impaired person they will get the options to call a volunteer or receive specialized help as seen in Figure 2.2. Since its launch in January 2015, over 6 million volunteers have signed up for the service[9], the vast majority being people who are willing to help others with simple tasks and have no formal training in doing so. This application is available for both Android and iOS.

Be My Eyes and similar solutions were discussed during the meetings with SONS. The following are observations extracted from their experiences using these applications:

- These applications are preferred for performing simple tasks such as reading the expiration date of medicine or food items and for finding objects in a room.

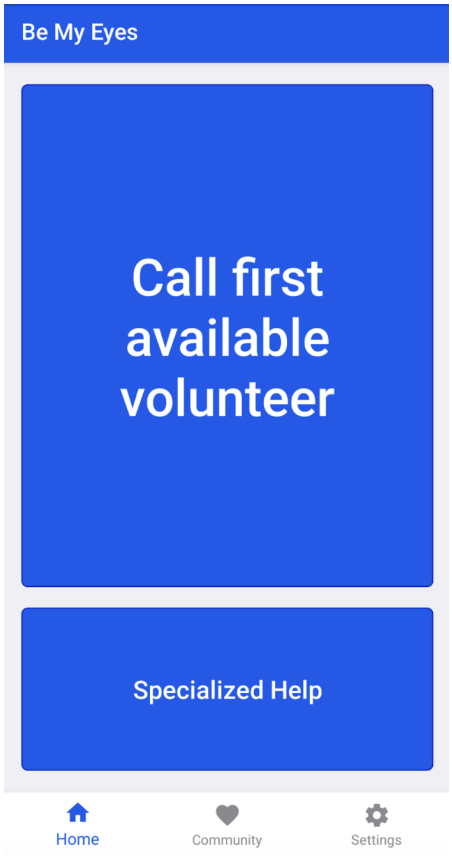

Figure 2.2 Be My Eyes main page for visually impaired users, obtained from [10].

- Their use on the street as an aid for navigation is not preferred since the instructions from the volunteer can be confusing and the latency of the call often makes it difficult for the user to react to navigation instructions promptly.
- For applications similar to Be My Eyes, the user experience can be inconsistent. Sometimes, the user requesting aid is willing to have a social element to the call, while other times, they require help quickly and the volunteer is more interested in establishing rapport first.
- For tasks where sensitive information is involved, such as making an online payment or reading statements on a bank bill, these applications are not preferred unless the person providing assistance is a close friend or a relative.
- A video call through a mainstream application such as Skype or WhatsApp is the preferred option for visually impaired users when they require help. Be My Eyes and similar applications are the second option in case they do not count on any acquaintance who can assist them at that particular moment or they do not wish to bother them with very simple tasks.
- The lack of a video resolution setting can result in quick consumption of mobile data. A user from SONS reported that even though a WiFi network was available and their device was connected to it the application consumed their mobile data for an unknown reason.

2.3 Applications for Navigation

Visually impaired users are able to navigate using applications such as Google Maps and Můj vlak with the help of the screen reading features of their device. They are able to create a schedule for the public transport routes that they will use in advance and efficiently navigate in an urban environment. They are also frequently used for finding their way across short distances in an environment that is new to them by switching to the pedestrian mode of these types of applications. During our meetings, we were able to capture the following insights:

- A sequence of navigation steps is usually created at the start of the day with the arrival times and locations of public transport services.
- Any disruption of arrival times such as delays or emergency maintenance on tram lines severely affects the user's ability to navigate since these changes are often posted in written form physically on the public transport stop but they are not broadcast proactively in most applications.
- In spaces where multiple public transport vehicles are coming at the same time, the probability of boarding a different one than what was planned is very high; this is something that most current applications cannot prevent. An application called Tramoji does provide the option to announce the order in which public transports are arriving; however, this application is no longer usable.
- In most cases, the layout of the application is not optimized for accessibility features resulting in the user having to go through many elements that are visually hidden while using features such as Talkback.
- Some applications use dynamic layouts that are very eye-catching but prove to be very difficult when trying to navigate properly using accessibility features, this is caused by the interactable elements being reordered in the screen after some user input.
- While a pedestrian mode is included in most navigation apps, important landmarks or obstacles on the street are not marked, which causes problems when users are trying to navigate an unfamiliar environment.
- Some applications have layouts that are very easy to navigate with accessibility features but the information they have about public transport routes is outdated or they are missing several features. There is simply no single application that combines a full set of functionalities, navigational layout and up-to-date information about public transport.
- When moving through a completely new environment, a navigation application is a must-have for a visually impaired user. If for any reason it is not possible to use such an application, the user more often than not will avoid approaching and exploring new environments.
- The features of the same application across different platforms can be different, which results in some devices being superior to others in terms of accessibility when executing the same application across different operating systems.
- The level of self-sufficiency when navigating varies depending on the person and the type of visual impairment is not an indicator of their ability to traverse their environment. Some people with partial sight are very reliant on navigation assistance and some fully blind individuals can navigate independently without substantial problems.

Chapter 3

Analysis and Requirements

We have researched different technologies available to visually impaired people and have gathered insights from their users. We will identify and summarize the shortcomings of the currently available solutions and present them in a clear and concise manner. We will also list the functional and non-functional requirements.

As discussed in the previous chapter, visually impaired users can make use of multiple tools and features when interacting with mobile and web applications. These options can be sufficient for the proper use of some applications, but they do not guarantee that the user experience will be positive or that in a situation where they need to get information about their environment, they will be able to get it in a reasonable amount of time.

3.1 Identifying Drawbacks

From this research, we have identified common drawbacks in the available applications. We have compiled them into a list of problems and we proposed possible solutions.

- **Problem 1:** List of components displayed on the screen can be hard to navigate using accessibility features or dynamic rendering may force the user to explore the entire component tree repeatedly after every action.

  Solution 1: Minimize the number of layout elements present in the application at any given time and avoid changing large portions of the layout frequently.

- **Problem 2:** Mobile data consumption is a limiting factor for the user of some applications when navigating without being connected to a WiFi network.

  Solution 2: Provide the user control over the data consumption of the application through settings such as being able to set the video stream quality without compromising the functionality of the application.

- **Problem 3:** Some applications only partially provide the features needed to complete certain tasks, forcing the user to switch between multiple applications on the same device.

  Solution 3: Include the most frequently used features into a single application, such as video calling, location sharing and image sharing.

- **Problem 4:** There is a discrepancy in the features offered by the same applications across multiple platforms, causing those users who own devices running a certain operating system to miss out on certain features.

  Solution 4: Develop using a single code base making use of technologies that can compile to different mobile operating systems and web applications, test the features on multiple devices and distribution channels.

- **Problem 5:** Application layouts break when increasing the font size, rendering some elements outside of the screen without enabling the option to scroll to see them.

  Solution 5: Create a responsive design that takes into account the changes to text font size and colors made by accessibility features and take special care not to include elements that can overlap each other or include text that becomes unreadable due to low contrast under these conditions.

- **Problem 6:** Timed actions are very difficult to respond to for a visually impaired user, as some text bubbles or push notifications may only appear for a few seconds at a time and the Talkback functionality of the device will not alert the user of their existence, or if they manage to find them, they may disappear by the time the screen reader finishes describing them.

  Solution 6: Do not include elements that require an action from the user and are only present for a few seconds at a time. Design the user experience in such a way that every step is sequential and there is little to no deviation in the flow of the user performing any action.

- **Problem 7:** Inputting text into an application can be a time-consuming process if no third-party devices are available since the current accessibility options are not very cooperative in terms of typing speed. Typing sensitive information can be problematic since a spelling mistake can be easily missed by a visually impaired user.

  Solution 7: Minimize the number of times the user needs to input text into the application. For cases in which a certain character string needs to be shared this process should be as automated as possible, either with the implementation of a request to the device's sharing functionality or by copying such string into the clipboard.

- **Problem 8:** Some accessibility features take control away from the user. For example, if the user is wearing headphones and the TalkBack feature detects that a call has been initiated, it may change the audio input source for the device. This behavior is frustrating for the user and severely hinders their ability to operate certain applications properly.

  Solution 8: Allow the user to select the video and audio sources that they want the application to use at any time. Store this configuration locally so they do not need to go into the settings every time.

- **Problem 9:** GPS coordinates in the pedestrian mode for navigation apps can be very imprecise in urban environments, especially when the user is surrounded by large structures such as tall buildings or is underground. If the requester is sharing their location in real time the assistant may provide an erroneous description of the route the requester must take due to these inaccuracies.

  Solution 9: When sharing the location of the requester include also the estimated precision of the GPS device, this would allow the helping party to make a better

description of the route that needs to be followed. Include also information from the device's accelerometer and compass. When the location is changing rapidly due to the device trying to calibrate these changes would allow the assistant to differentiate between real movements from the user and changes that happen due to the device's own inaccuracy.

3.2 Functional Requirements

During the meetings with SONS, the idea of a real-time assistance application for visually impaired users was brought up and through discussion about their needs and previous experiences we were able to write a requirement list. For defining the requirements, we selected the most commonly used features from different applications and proceeded to focus on those that were feasible to implement within the scope of this work.

- **Video and audio call:** The ability to establish an audio call for both parties and to stream video from the device of the visually impaired user to the fully sighted user.
- **Geographical information:** Sharing the location of the visually impaired user in real time, displaying their location in a map on the fully sighted user's side along with the estimated precision of the GPS coordinates.
- **Sensor information:** Streaming the acceleration and orientation of the visually impaired user's device in real time. This information is displayed to the fully sighted user to aid in the navigation assistance.
- **High definition picture capturing:** The ability for a high definition picture to be taken from the visually impaired user's device without compromising the integrity of the audio call. This picture is then sent to the fully sighted user.
- **Video quality control:** Selecting the quality of the video stream originating from the visually impaired user's device.
- **Camera switching:** Changing the input camera for the video stream.
- **Remote device control:** Allowing the fully sighted user to take pictures, change the quality of the video and switch the input camera on the visually impaired user's side.

3.3 Non-Functional Requirements

This list was extracted by discussing the shortcomings of currently available solutions as well as the resources that are currently allocated to SONS.

- **Low operational cost:** A common complaint from visually impaired users is that current solutions have a very high price point which is not sustainable for most users in this group. Similar solutions have been proposed previously to SONS, but the cost of hosting and development has rendered them prohibitively expensive. For this reason, keeping the infrastructural costs as low as possible is a top priority for this project.
- **Multi-platform support:** Visually impaired users make use of multiple devices to navigate the web and receive assistance. For this reason, supporting both Android and iOS is a must. Web browser support is also a requirement since the fully sighted user can better provide assistance through the use of a personal computer.

- **VoiceOver support:** The graphical elements of the application must be able to be discovered and interacted with while the VoiceOver functionality is active on any mobile device.
- **Cross-platform connectivity:** A visually impaired user may request help on a mobile device running the Android operating system while the fully sighted user provides help using a device running iOS. For this reason, the call functionality must be able to connect users on any combination of the supported devices.
- **Low latency audio call:** A visually impaired user navigating an urban environment needs instructions transmitted with very little delay in order to safely move through their environment. For this reason, keeping the latency on calls as low as possible is crucial.

In order to satisfy these requirements, a peer-to-peer architecture was considered since the beginning. This would ensure that a back-end service would not need to route the heavy audio and video call traffic, reducing the operational cost and the latency of the calls.

Chapter 4

Exploring Potential Solutions

Multiple potential solutions were researched and tested in order to select the most adequate framework and language for this task. We will explore potential solutions and report findings gathered through experimentation and fast prototyping.

We will describe the frameworks and tools explored during the iterative development of this solution. Different technologies were tested by coding a fast prototype that would initiate a video stream from one device to another. During this exploration we encountered problems that would render each possibility unusable, an overview or these solutions and a brief description of such problems is provided in this chapter. Thanks to the lessons learned during this experimentation phase we were able to make an informed decision about the technology with which we would implement the final solution.

4.1 Ionic Framework

Ionic is an open source user interface toolkit for building high quality cross-platform mobile apps writing a single code base using the React framework [11]. This framework was the first candidate for development and a partially working prototype was developed using it. The reasons for this were cross-platform support, familiarity with the technology, offering of pre-built UI components with accessibility features included and the existence of libraries that interact with the hardware of the device.

Ionic has support for both JavaScript and TypeScript programming languages. It has been used to build applications for many large companies including Panda Express, CAT and Cisco [12]. Support for compiling to PWAs (Progressive Web Applications) is also included, which would be useful if the helping party prefers them over traditional web browser pages.

This tool was ultimately dropped due to the lack of support for camera streaming across the supported platforms. Ionic exposes a Camera API for taking pictures and recording video but does not provide the ability to stream from this video source. The lack of documentation and frequently outdated plugins were reasons for abandoning this approach.

Code listing 4.1 Example of the *accessible* attribute.

```
<View accessible={true}>
  <Text>This is some text</Text>
  <Text>And this is more text</Text>
</View>
```

4.2 React Native

React native is an open-source JavaScript framework for building native mobile applications on both Android and iOS. It has an ongoing effort to support compilation to a web environment through a community-led plugin [13]. This framework was considered due to the multi-platform support, large community backing and familiarity with the technology.

React native has tools that provide support for accessibility features such as the *accessible* property, see Code listing 4.1. The inclusion of this feature will result in both *Text* elements being recognized as a singular element by the voice-over functionality of the device by grouping into a single node using the *View* element. This behavior is desired when the user interface contains multiple separate elements that combine to form a single cohesive section but which do not need to be explored individually by the accessibility features.

A proof-of-concept application was built using this framework, but its use was scrapped due to the inconsistencies between functionality in native and web environments. The documentation was also a factor in this decision. Since most of the plugins are written by community members, the documentation is frequently fragmented and incomplete for several components.

4.3 Apache Cordova

Apache Cordova is an open-source mobile development framework [14]. This framework was considered due to the support of multiple community-created plugins with centralized documentation pages and the comprehensive list of supported features across different platforms.

These are the plugins that the Ionic framework uses to access the device's hardware [15]. Many of the plugins written in Cordova are not supported in Ionic, which is another reason why developing the solution using this framework was considered.

A single-page application meant to test the core functionalities was written using this framework. Unfortunately, this exploratory development did not yield the expected results due to problems with the main camera plugin and the lack of support and documentation on it. Setting up the development environment for this framework also proved to be a largely difficult process with many features such as hot reload not being supported by default and most of the plugins that offer these functionalities are incompatible with the latest version of the framework.

4.4 PeerJS

PeerJS is a library that wraps the browser's WebRTC implementation to provide a complete, configurable, and easy-to-use peer-to-peer connection API [16]. It provides func-

tionality to make video and audio calls, it also creates media connections through which information can be exchanged between peers.

This library is at the core of the application. It was selected early on in development due to it being supported by any framework that runs on JavaScript and providing the basic functionalities needed to satisfy the requirements.

By establishing a peer-to-peer connection, a low delay between both parties can be ensured since the exchanged data do not need to go through a third-party server. The only extra setup needed is an instance of the PeerJS server if more control over availability and peer id generation is desired. This is however not a strict requirement since a free public server is provided by the developers.

Another library that mirrors the implementation of this library and provides the same features was used for the final solution.

Chapter 5
Final Solution

What set of technologies provides the support and development tools necessary for this application? After multiple iterations we decided to settle on the Flutter framework, the reasons why this technology was considered appropriate are described in this chapter.

In this chapter, we will describe the technology chosen for the final implementation of the application, its advantages and disadvantages, and elaborate a description of the functionality with code examples. Describe shortcomings and unexpected behaviors encountered during the development process and for which there is little documentation available. We will also illustrate the architecture for the components of the final solution and describe the tools used to deploy it into production. The source code for this solution is available in the attachment to this thesis.

5.1 Dart

Dart is a programming language designed by Lars Bak, the Danish computer programmer responsible for the development of the V8 JavaScript engine which is used in Chromium-based web browsers and independent projects such as Electron [17].

Dart is an object- and class-oriented language with integrated garbage collection and syntax of the style of C. It is able to compile to both machine code and JavaScript. It supports the usual features of an object-oriented language such as interfaces, abstract classes and mixins while also providing integration for generics and type inference [18]. The example in Code listing 5.1 showcases how Dart is able to support dynamic types without compromising the soundness of the program due to a type mismatch that can be caught during compile time. This feature of the programming language is relevant due to my own personal experience using programming languages that are not strictly typed, by supporting dynamic typing and also checking for errors in compile time Dart offers the flexibility of typing that I am used to while also providing soundness in the code.

The function *printList* has defined as a single parameter an object of type *List<String>* while the variable *names* has been assigned an implicit type of *List<Dynamic>* by the compiler due to the lack of information provided to the analyzer to assign a more specific type. This results in the error shown in Code listing 5.2.

This error occurs due to an unsound implicit cast from the implicit type *List<Dynamic>* of *names* to the explicit type *List<String>* of the parameter in *printList*. This example

```
void printList(List<String> list) => print(list);

  void buildList() {
    List names = [];
    names.add('Ivan');
    names.add('Erik');

    printList(names);
  }
```

Code listing 5.1 An example to showcase type mismatch handling in Dart.

```
The argument type 'List<dynamic>' can't be assigned to the
parameter type 'List<String>'.
```

Code listing 5.2 A type mismatch error message in Dart.

showcases how Dart is able to support dynamic types without compromising the soundness of the program due to a type mismatch that can be caught during compile time.

Dart was first unveiled in 2011 during the GOTO conference in Aarhaus, Denmark [19]. It originally had a mixed reception, being frequently criticized by some developers for fragmenting the web due to the introduction of yet another tool for building websites. This sentiment resulted from to the planned inclusion of a Dart virtual machine in Chrome. These plans were eventually canceled in 2015 and instead, Dart switched focus to supporting compilation to JavaScript [20]. This change was brought about due to the feedback of multiple developers stating that they loved working with Dart and its libraries and tools but still opted for tools that compile to JavaScript when they deploy to the web. The European Computer Manufacturers Association formed the technical committee TC52, which seeks to perform standardization work for the Dart programming language [21]. Thanks in part to the standardization efforts the usage of the language grew significantly. The ability to compile to standard JavaScript as also included which ensures that it can work in any modern browser with JavaScript support.

5.2 Flutter

Flutter is an open-source framework by Google used to develop natively compiled cross-platform applications for iOS, Android, macOS, Linux, Windows, Google Fuchsia and the web making use of a single codebase. It consists of a thin layer of C and C++ code and implements the majority of its components in Dart [22].

5.2.1 History

Flutter was first mentioned in 2015 during the Dart developer summit. It was originally called “Sky” and was showcased on a real Android device. It was compiled into a native Android application without any Java code and it supported animations, multi-touch, network connectivity and user interface elements such as lists, text inputs and drawers [23]. It started development with the following goals: Performance, to provide a responsive user experience ideally running at a refresh rate of 120Hz. Platform agnosticism, being

able to be ported into Android, iOS and more platforms. Full access, do not restrict any device permission from the user, build the necessary implementations so that the developer can access any tool and sensor that the device has. Continuous deployment, streamline the process needed to update previous versions of the application taking as a guideline the typical web development pipeline. Rich & Flexible Layout & Painting, do not constrict the developer to predefined layouts, add the support for the developer to extend or build their own layouts as it is needed to achieve any configuration.

5.2.2 Support

During the following years Flutter grew in popularity and it is currently on version 3.7 with the support for the platforms listed in Table 5.1.

Table 5.1 Supported platforms for Flutter deployment. **Supported:** Platforms tested by Google automatically by continuous integration testing tools. **Best effort:** Platforms supported only by coding practices tested on an ad-hoc basis. **Unsupported:** Platforms that are neither tested nor supported. [24]

Platform version	Supported	Best effort	Unsupported
Android SDK	21-30	19-20	≤ 18
iOS	14-15	11-13	≤ 10, arm7v 32-bit
Linux Debian	10-11	≤ 9	
Linux Ubuntu	18.04 LTS	20.04	any 32-bit
macOS	Monterey (12+)	Mojave (10.14) to Big Sur (11)	High Sierra (≤ 10.13)
Chrome (web)	latest 2 releases	≥ 96	
Firefox (web)	latest 2 releases	≥ 99	
Safari (web)	latest 2 releases		
Edge (web)		≥ 96	
Windows	10	7 & 8	$\leq$ Vista, any 32-bit

Flutter offers both ad-hoc and tool-tested support for multiple platforms. For the purposes of this project, we will focus only on the web, Android SDK and iOS platforms. Flutter implements Dart's package manager and software repository, this resource functions as a centralized repository for user-created packages and plugins that are specific to Flutter. There are over 17,000 packages that support the Android, iOS and Web platforms available for development.

5.2.3 The Widget Component

The user interface in Flutter is built upon a basic component called a widget. These widgets take inspiration from React's components. Each widget describes how its own view should look like with its current configuration and state. Upon a state change, each widget will rebuild its own portion of the view. The framework compares the current description of a widget with the previous description in order to determine the minimal amount of changes needed in the render tree to reflect the state change.

Widgets in Flutter are handled internally using a tree structure analogous to the DOM implementation of web pages. With every new widget added to the application, a new insertion to the tree will be created. These widgets are divided into two categories:

Stateless widgets: These widgets do not contain an internal state, their render method is only run once upon creation. Data inside this widget is immutable and will

```
// This widget will render a simple text component
class TitleText extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return const Text('This is the title!');
  }
}
```

Code listing 5.3 A stateless text widget in Flutter.

only change when a new instance of the widget is created with a different configuration. They are commonly used for elements of the user interface that will not change during the life cycle of the application. An example is shown in Code listing 5.3.

Stateful widgets: These widgets are dynamic and contain a mutable internal state that will affect how the widget is presented. They can be modified without the need to be initialized and inserted into the widget tree again, the different stages and methods that a stateful widget has can be observed in Figure 5.1.

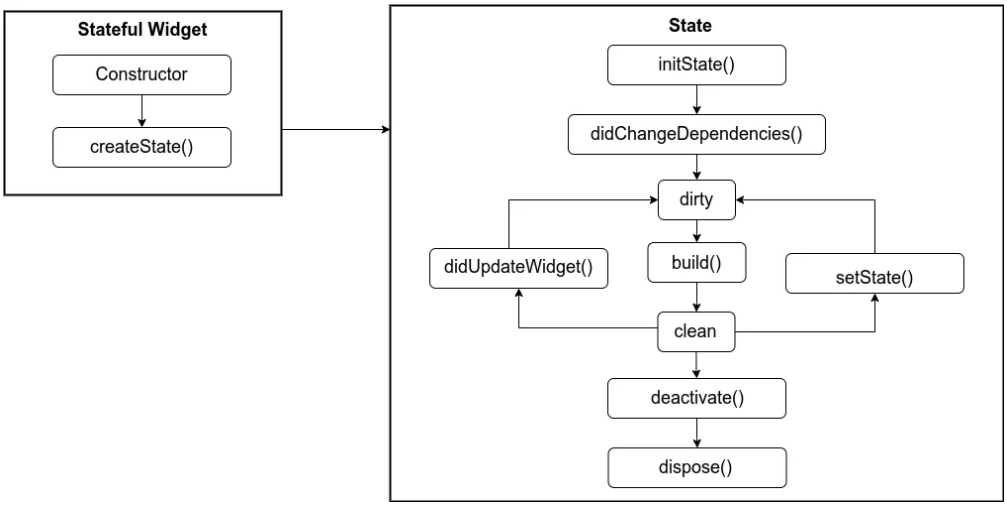

Figure 5.1 Stateful widget life cycle methods [25].

Code listing 5.4 contains the code for a minimal stateful widget that counts the number of items the user has pressed a button and displays the total amount of presses.

The Code listing 5.4 will render the page shown in Figure 5.2.

In this example, the state contains the *_counter* variable. The *late* keyword of this variable indicates to flutter that it will not have a value assigned during the declaration and a value will be assigned to it later. It is the responsibility of the developer to make sure that any *late* variable has a value before being used.

The *initState* function will be executed only once after the widget has been inserted into the widget tree and it will initialize the *counter* value of the state to zero. Since we know that *initState* will always execute before *build*, then we can assure that the variable has a value assigned before its usage and no error will be thrown. When the button is pressed, the *setState* function is called and the framework performs the necessary checks to render the component again with the minimal amount of changes needed to reflect the new state. In this case, it will be an increment to the number in our text widget.

```
import 'package:flutter/material.dart';

class CounterWidget extends StatefulWidget {
  const CounterWidget({super.key});

  @override
  State<CounterWidget> createState() => _CounterWidgetState();
}

class _CounterWidgetState extends State<CounterWidget> {
  late int _counter;
  @override
  void initState() {
    super.initState();
    _counter = 0;
  }

  @override
  Widget build(BuildContext context) {
    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Text('Times clicked: $_counter'),
        ElevatedButton(
            onPressed: () => setState(() {
                  _counter++;
                }),
            child: const Text('Click me!'))
      ],
    );
  }
}
```

Code listing 5.4 A stateful counter widget in Flutter

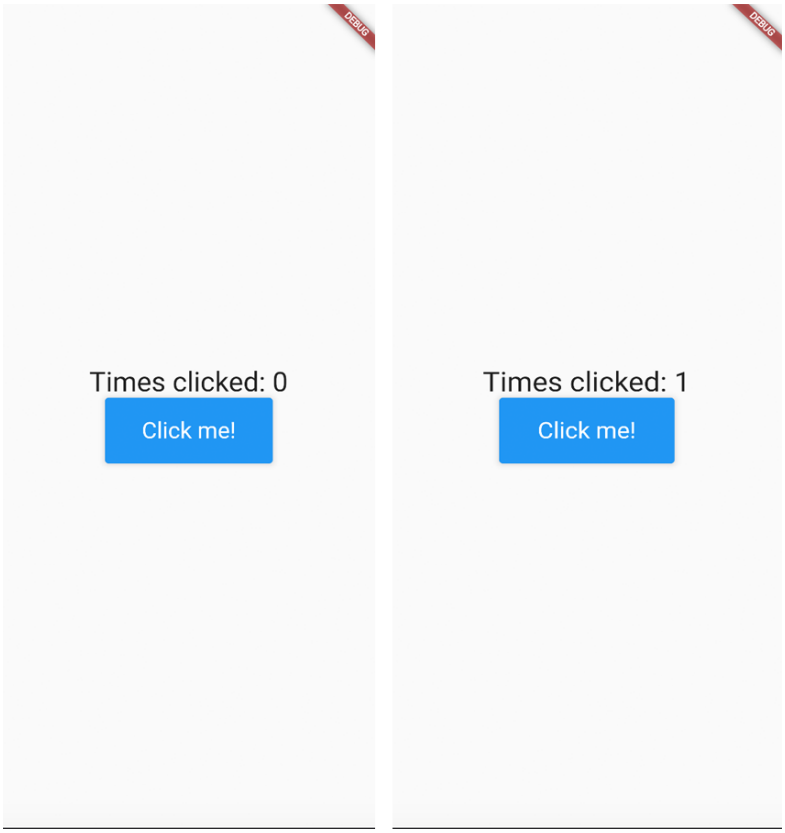

Figure 5.2 Stateful widget example.

5.2.4 State Management

Flutter supports multiple state management approaches starting with the widget-specific *setState* function which will manipulate the internal state of the widget. For more complicated cases where a widget needs to affect a different widget due to internal changes, a more elaborate approach is required. The tree structure of widgets makes it difficult for two components that are on a similar level to communicate with each other if the common ancestor they share is very high up in the tree structure. In order to deal with this problem, Flutter supports a variety of options for handling state management.

For this solution, the *ChangeNotifier* approach was chosen. *ChangeNotifier* is a class included in the Flutter SDK, which provides the ability to broadcast change notifications to listener classes. This behavior is an implementation of the *observable* approach in which changes are triggered every time a modification or instantiating of a certain element occurs. Code listing 5.5 contains the code for a simple *ChangeNotifier* implementation.

The *CountModel* class extends *ChangeNotifier* and contains the representation of the state for the application, in this case a counter of a button press. The functionality obtained from the *ChangeNotifier* class is the *notifyListeners* function. When called, it will notify all registered observers of any resource listed in the *CountModel* class that a state change has happened and a change in the user interface may be needed. For a widget to be able to access this notifier, it must be within the scope of a parent widget that contains a *ChangeNotifierProvider* element as exemplified in Code listing 5.6.

ChangeNotifierProvider provides an instance of a class that extends *ChangeNotifier*

```
class CountModel extends ChangeNotifier {
  int _pressCount = 0;

  int get pressCount => _pressCount;

  void increasePressCount() {
    _pressCount++;
    notifyListeners();
  }
}
```

Code listing 5.5 Example with *notifyListeners()* on a ChangeNotifier.

```
class AppContainer extends StatelessWidget {
  const AppContainer({super.key});

  @override
  Widget build(BuildContext context) {
    return ChangeNotifierProvider(
      create: (context) => CountModel(),
      child: const CounterWidget(),
    );
  }
}
```

Code listing 5.6 Scoping components inside a *ChangeNotifierProvider* element.

to the widgets that are its descendants in the widget tree. Finally the *pressCount* value and *increasePressCount* functions can be accessed from the widget context by using the code from Code listing 5.7.

The function *context.watch* will obtain a value from the nearest ancestor provider of the specified type and subscribe to it. It will make the widget rebuild every time there is a change in the value of *pressCount*.

The function *context.read* will also obtain the value from the nearest ancestor provider of the specified type but it will not trigger a rebuild when the value changes, in this case it will enable the *_CounterWidgetState* component to access the *increasePressCount* function to change the *pressCount* value.

In this example there is only one widget observing the state of *CountModel* but the method for exposing that resource to more widgets is the same as long as they are also an ancestor of the *AppContainer* widget in order to be able to access the *CountModel* from the context.

One important observation is that *context.watch* will listen for changes on the entire object leading to potentially unnecessary rebuild cycles for widgets. This is usually not a problem for small applications that require only a single *ChangeNotifier* but for more complicated implementations, we can make use of the *context.select* function:

```
int pressCount = context.select((CountModel m) => m.pressCount);
```

The function *context.select* will trigger the build function in similar conditions to *context.watch* with the exemption that it will only listen to a certain attribute of the *ChangeNotifier* state, ignoring changes on attributes that are not relevant for the widget.

```
class _CounterWidgetState extends State<CounterWidget> {
  @override
  Widget build(BuildContext context) {
    int pressCount = context.watch<CountModel>().pressCount;

    return Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        Text('Times clicked: $pressCount'),
        ElevatedButton(
            onPressed: () {
              context.read<CountModel>().increasePressCount();
            },
            child: const Text('Click me!'))
      ],
    );
  }
}
```

Code listing 5.7 Accessing functions from *ChangeNotifierProvider* in a different widget.

```
    String preferredId = 'my-name';
    Peer peer = Peer(id: preferredId);

    peer.on('open').listen((peerId) {
      print('The id I got from the server is $peerId');
      peer.call(otherPeerId, mediaStream);
    });
```

Code listing 5.8 Initializing a peer.

5.2.5 PeerDart

The application relies on PeerDart to provide the core functionality. PeerDart is a plugin that provides a peer-to-peer API implemented using WebRTC with support for data and media streams [26].

The plugin mirrors the implementation of PeerJS, a library written in JavaScript that encapsulates the browser's WebRTC implementation and provides a peer-to-peer connection API [16].

The connection is made on the basis of a unique peer id. This id will be used to identify the device running the library and needs to be shared in order to start a call. The basic code to start a call is listed in Code listing 5.8.

The *preferredId* is the peer id that the peer will request from the server. If no id is specified, the server will generate a random string as an id. Once the peer triggers the *open* callback, it is ready to start a call with a *String* that corresponds to the id of the other party and a *MediaStream* containing at least one audio or video track.

PeerJS exposes a public cloud service that the peer will connect to if no *host* and *key* options are provided during instantiating. For the purposes of this implementation, the public server is being used. For extra control, it is also possible to host a custom server by cloning and modifying the *peerjs-server* repository [27].

5.3 Architecture Overview

Making use of the features the chosen technology offers, we have been able to implement a solution that covers all of the basic features discussed during our meetings with SONS. With the current implementation, we have been able to avoid paid services and the application is currently publicly accessible with zero operational cost.

The architecture of the solution is described in Figure 5.3. The Peer Server is responsible for handling the initial connection between peers. After the connection has been established, the server stops hosting data about the peer objects and all future traffic will be done directly between devices without passing through the server. Currently, the only data persistence occurs locally in the device of the user and is used to store setting preferences.

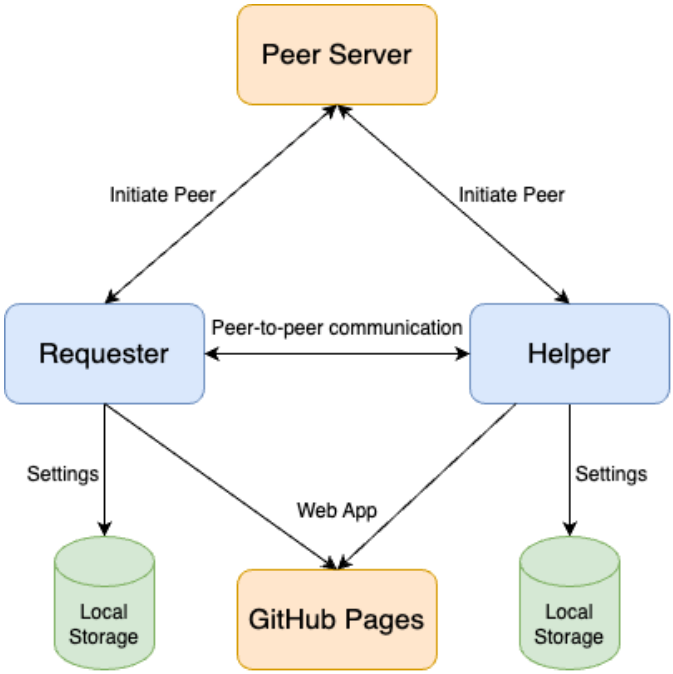

Figure 5.3 Architecture of the application.

5.4 Implementation

The final implementation was developed using Flutter with support for web, Android and iOS deployment. The application makes use of multiple community-created plugins and a peer-to-peer library for handling video calls and data transfer between devices. Through testing and developing we were able to discover discrepancies in behaviour between emulators and real devices. Multiple drawbacks for solving this problems arose because of the lack of documentation for different platforms. Nonetheless the development yielded a product that satisfactory covers the majority of requirements and was tested with visually impaired users during our meetings with SONS.

The application is divided into two different flows differentiated by the functionality present in each of them and being encompassed by one of two implemented *ChangeNotifier*

providers. On the homepage, the application presents the user with buttons to initiate either flow or adjust the application settings, as shown in Figure 5.4.

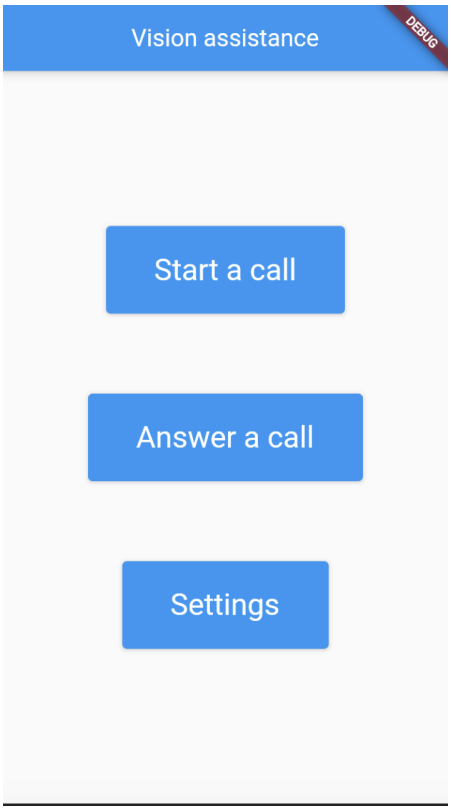

Figure 5.4 Main page of the application.

In the settings section, the user can input a preferred call id; this will be explained later on. It is also possible to select the default camera that the device will use when starting a video call, as shown in Figure 5.5.

When the save button is pressed or a default camera is selected, a message on the bottom of the page appears and a text-to-speech voice alerts the user of the change. The settings are persisted using the *LocalStorage* plugin [28].With this plugin a JSON file-based storage is created and the settings can be persisted between use sessions in the same device.

5.5 The Requester

The requester side of the application is composed of the widgets that are nested inside the scope of the *RequesterProvider ChangeNotifier*. This class handles business logic and encapsulates the resources present in Code listing 5.9.

Now we will provide a description for all the resources included in this class.

_localVideoStream: A *MediaStream* obtained from the camera of the device. This resource will be streamed on a one-way call to the assistant party.

_localAudioStream: A *MediaStream* obtained from the audio input of the device. It will be used for the bi-directional audio call.

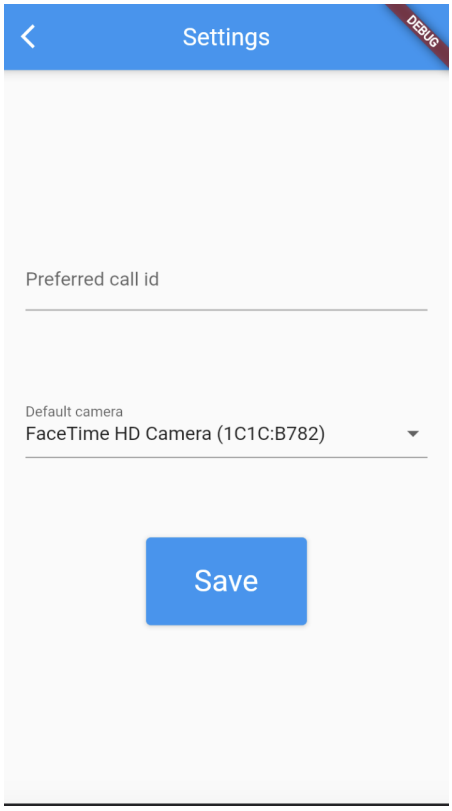

Figure 5.5 Settings page.

_remoteAudioStream: A *MediaStream* obtained via streaming from the device of the assistant party. It contains only audio tracks.

_geoStream: A *StreamSubscription* generated by the *geolocator* plugin [29]. It will be used to obtain the coordinates of the device and send them to the assistant party after a movement threshold has been reached.

_speedStream: A *StreamSubscription* generated by the *sensors_plus* plugin [30]. This resource will periodically access the device accelerometer to calculate the current speed and stream it to the assistant party.

_localStorage: A resource provided by the *localstorage* plugin [28]. It is used to persist information in JSON format locally in the device.

_dataConnection: A *DataConnection* object provided by the *peerdart* plugin [26]. It is used to send JSON messages between both parties.

_videoConnection: A *MediaConnection* provided by the *peerdart* plugin. It will be the connection used to stream the *_localVideoStream* resource to the assistant party.

_audioConnection: A *MediaConnection* provided by the *peerdart* plugin. It will handle the bi-directional audio connection between the requester and the assistant party.

Every resource implements the *ResourceState* class shown in Code listing 5.10.

This ensures that every resource exposes an usable Boolean indicator for usability and implements internal logic to dispose of the resource since some of the resources need to be stopped and disposed while others can be disposed without stopping or canceling.

The basic flow of the connection process for a call is represented in Figure 5.6.

The user interface for the requester is generated according to the status of the call.

```
  // Media and data streams
  final MediaStreamResource _localVideoStream =
      MediaStreamResource(false, null);
  final MediaStreamResource _localAudioStream =
      MediaStreamResource(false, null);
  final MediaStreamResource _remoteAudioStream =
      MediaStreamResource(false, null);
  final StreamSubscriptionResource _geoStream =
      StreamSubscriptionResource(false, null);
  final StreamSubscriptionResource _speedStream =
      StreamSubscriptionResource(false, null);
  final LocalStorageResource _localStorage =
      LocalStorageResource(false, null);

  // Data connections
  final DataConnectionResource _dataConnection =
      DataConnectionResource(false, null);
  final MediaConnectionResource _videoConnection =
      MediaConnectionResource(false, null);
  final MediaConnectionResource _audioConnection =
      MediaConnectionResource(false, null);
```

Code listing 5.9 Resources of *RequesterProvider*.

```
abstract class ResourceState {
  bool usable = false;
  Future disposeResource();
}
```

Code listing 5.10 *ResourceState* class.

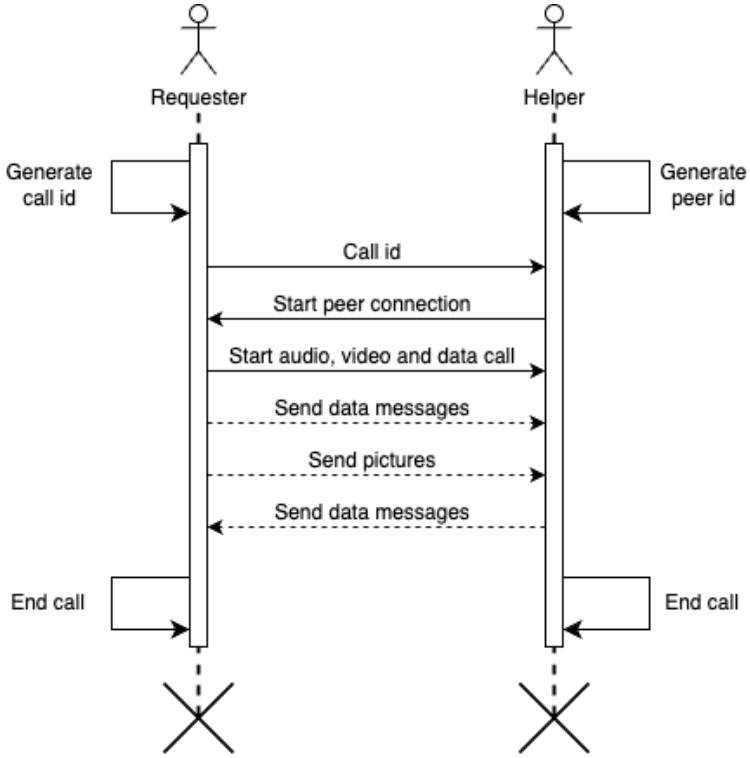

Figure 5.6 Sequence diagram of the connection process.

```
enum RequesterCallStatus {
  startingResources,
  resourcesStarted,
  ongoingCall,
  callEnded
}
```

Code listing 5.11 Call status enum.

This status is represented at a code level by the enum in Code listing 5.11.

5.5.1 Starting Resources

When the *RequesterCallStatus* value in *RequesterProvider* is *startingResources*, the application will render the view of Figure 5.7.

While on this page the application will make a text-to-speech announcement and initialize the *_localStorage*, *_localVideoStream*, *_localAudioStream*, *_geoStream*, *_speedStream*, *_peerResource* resources using the local sources in the device. After the resources are initialized and the peer has connected to the server and is in an *open* state the *RequesterCallStatus* will be changed to *resourcesStarted*.

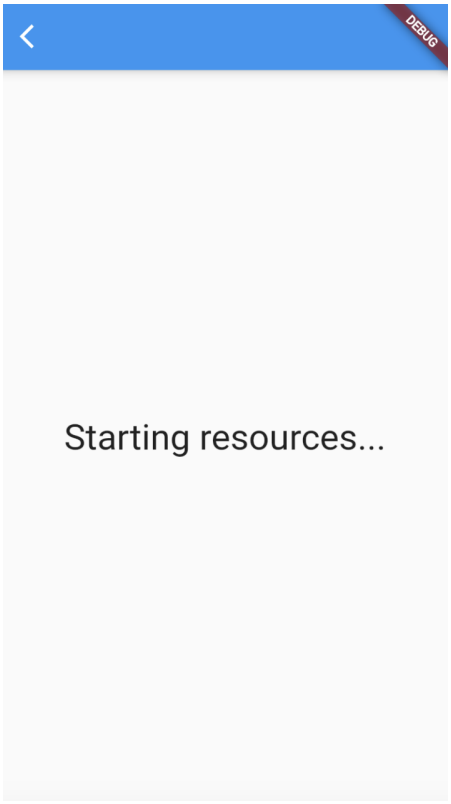

Figure 5.7 Starting resources page.

5.5.2 Resources Started

Figure 5.8 will be rendered if the *RequesterCallStatus* value in *RequesterProvider* is *resourcesStarted*.

Here, the local peer id is displayed and the options to copy the code to clipboard and share the code are presented. When pressing *Share code*, a call will be made to the *Share plugin*[31] which will open the system share dialog on the corresponding platform. On a mobile device, the code can be shared via any messaging app such as Messenger, WhatsApp, Telegram or SMS from this menu.

When a data connection has been established, the *RequesterCallStatus* will be changed to *ongoingCall*.

5.5.3 Ongoing Call

Figure 5.9 will be rendered if the *RequesterCallStatus* value in *RequesterProvider* is *ongoingCall*.

On this page, four actions are available for the requester:

Switch camera: This option cycles through the available cameras of the device. When clicked it will stop and dispose the current local video stream and start a new video stream with the next camera using the logic in Code listing 5.12. Using text-to-speech, the change will be notified to the requester and the label of the button will be changed to display the label of the newly active camera.

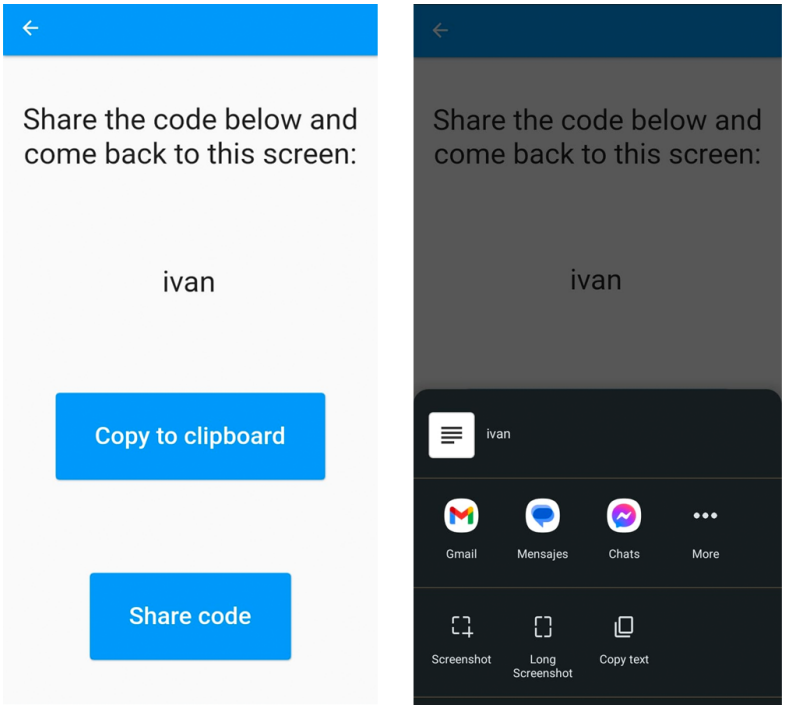

Figure 5.8 Resources started page and share code menu.

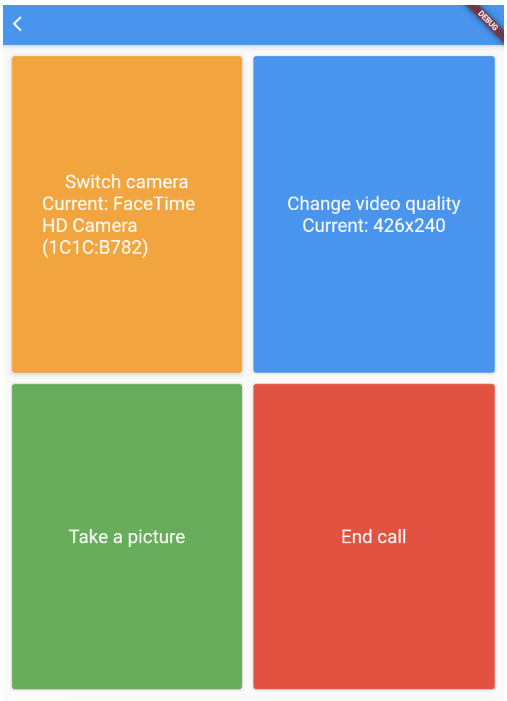

Figure 5.9 Ongoing call page.

Change video quality: When pressed, the local video stream will be stopped and a new video stream will be started with a different resolution. The two currently supported

```
void changeCamera() async {
    if (_cameras.length <= 1) {
      TextSpeak().speak('Only one camera available');
      return;
    }

    MediaDeviceInfo activeCamera =
        _cameras.firstWhere((MediaDeviceInfo deviceInfo) {
      return deviceInfo.label == _activeCameraLabel;
    });

    _activeCameraIndex = (_cameras.indexOf(activeCamera) + 1)
        % _cameras.length;
    _activeCameraLabel = _cameras[_activeCameraIndex].label;
    TextSpeak().speak('Switched camera to $_activeCameraLabel');
    _restartCameraFeed();
  }
```

Code listing 5.12 Code for taking a picture without opening the camera app.

resolutions are 426x240 and 1280x720 pixels. After the change is complete, the application will use text-to-speech to notify the user if the device is currently streaming in high or low resolution. Allowing control for this stream aids in addressing the data consumption of the requester user as mentioned in Chapter 3.

Take a picture: This button will pause the video stream and initiate a call to the camera plugin [32] to take a picture from the back camera of the device. The picture will be taken in medium resolution without opening the camera application or stopping the audio call. This picture is stored in memory and converted to a base-64 string which will be sent to the assistant party. After this picture is sent text-to-speech will be used to notify the requester user. The function to achieve this is in Code listing 5.13.

End call: This button will end the audio, video and data connection to the requester party and set the *RequesterCallStatus* value in *RequesterProvider* to *callEnded*.

5.5.4 Call Ended

When the status has been set to *callEnded*, the view in Figure 5.10 will be rendered. All of the resources in *RequesterProvider* will be disposed of when this page is rendered. If another call is started, the application will render the view in Figure 5.7 and the process for starting a call will be repeated.

5.6 The Assistant

The assistant side of the application is accessible by selecting the *Answer a call* button, see Figure 5.4. It is defined by the widgets that are within the scope of the *HelperProvider* class which extends *ChangeNotifier*. This class contains the resources mentioned in Code listing 5.14.

We will now provide a description about the resources included in the *HelperProvider* class.

```
void takePicture() async {
  try {
    TextSpeak().speak('Taking a picture');
    await _videoConnection.disposeResource();
    await _localVideoStream.disposeResource();
    notifyListeners();
    await Future.delayed(const Duration(milliseconds: 100));

    final cameras = await availableCameras();
    final backFacingCamera = cameras.firstWhere(
      (camera) =>
          camera.lensDirection == CameraLensDirection.back,
      orElse: () => cameras.first,
    );

    CameraController controller =
        CameraController(backFacingCamera,
        ResolutionPreset.medium);
    await controller.initialize();
    await Future.delayed(const Duration(milliseconds: 300));
    final XFile picture = await controller.takePicture();
    String base64Image = base64Encode(
      await picture.readAsBytes());
    await controller.dispose();

    await _dataConnection.connection
        ?.send(getEncodablePictureMessage(base64Image));

    await _restartCameraFeed();
    await controller.dispose();
    TextSpeak().speak('Picture taken');
  } catch (err) {
    TextSpeak().speak('Could not take a picture');
  }
}
```

Code listing 5.13 Take a picture code.

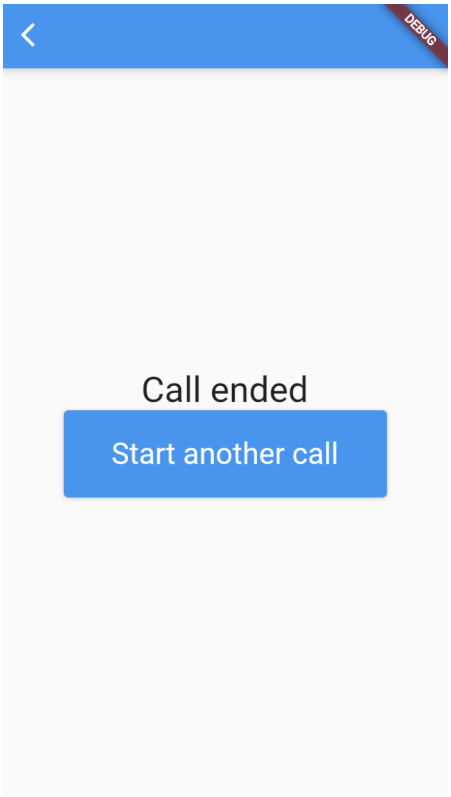

Figure 5.10 Call ended page for the requester.

_localAudioStream: The *MediaStream* obtained from the device's audio input. It will be used to answer a call started by the requester side.

_remoteVideoStream: A *MediaStream* obtained remotely from the requester's device. It will be displayed when a call is ongoing, this object can be set many times during the course of the call since a change in camera or resolution of the video in the requester side will produce a new *MediaStream*.

_remoteAudioStream: A *MediaStream* obtained remotely from the requester's device. It contains only one audio track and no video tracks, this object is immutable during the duration of the call.

_dataConnection: The *DataConnection* object used to send and receive messages between the requester and assistant devices during the call.

_videoConnection: The *MediaConnection* object that provides the *_localAudioStream MediaStream*. This connection may be disposed and started again multiple times in a single call.

_audioConnection: The *MediaConnection* object that provides the *audioConnection MediaStream*. This connection will remain active throughout the duration of the call, if at any point this connection stops the call will be considered closed.

_peerResource: A *Peer* that will be used to connect the assistant device with the requester device. The internal id for this peer is provided by the server and remains unknown for both parties for the duration of the call.

The user interface for the provider is generated according to the status of the call which is represented by the enum in Code listing 5.15.

```
  HelperProvider() {
    _resources = [
      _remoteVideoStream,
      _remoteAudioStream,
      _videoConnection,
      _dataConnection,
      _audioConnection,
      _localAudioStream,
      _peerResource,
    ];
  }
```

Code listing 5.14 Resources of the *HelperProvider* class.

```
enum HelperCallStatus {
  startingResources,
  waitingPartnerId,
  connectingToPartner,
  ongoingCall,
  callEnded
}
```

Code listing 5.15 Requester call status enum.

5.6.1 Starting Resources

When *HelperCallStatus* is set to *startingResources*, the application will request access to the user's audio input device and the *HelperProvider* class will initialize the resources in Code listing 5.14. The rendered view will be the same as in Figure 5.7.

After the resources are initialized and the *_peerResource* in Code listing 5.14 has been connected to the server and is in an active state, the *HelperCallStatus* will be set to *waitingPartnerId*.

5.6.2 Connecting to the Requester

This part of the flow starts when the *HelperCallStatus* is set to *waitingPartnerId*, the page in Figure 5.11. After the assistant has received the code from the requester party and it has been submitted on this page, a transitional state will be triggered by setting *HelperCallStatus* to *connectingToPartner*, during which Figure 5.12 will be rendered.

While Figure 5.12 is displayed, the assistant peer is establishing a data connection to the requester peer using the code that was previously shared. After a data connection has been established, audio and video connections will be started from the requester side and after they have been started, *HelperCallStatus* will be set to *ongoingCall*. The audio connection is separate from the video connection in order to achieve a constant stream of audio regardless of the state of the camera on the requester side, this behavior is useful since it allows both parties to communicate during the picture taking process.

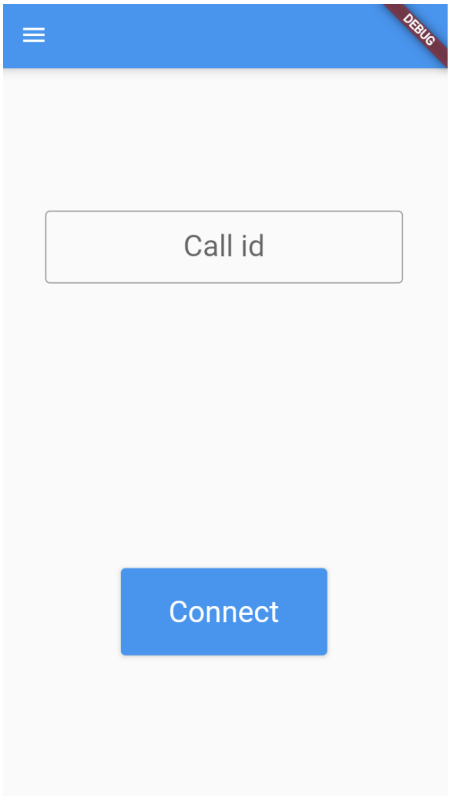

Figure 5.11 Waiting for partner id page.

5.6.3 Ongoing Call

When *HelperCallStatus* is set to *ongoingCall*, the view from Figure 5.13 will be rendered.

When this page is reached, a live audio, video and data connection is ongoing. The screen will be divided into two sections, the first section will stick to the left side of the screen, if the device used by the assistant is wider than its height. And it will otherwise be positioned on the top half of the screen. The first section constantly displays a video feed from the requester side with the following controls:

Rotate video: Rotates the video feed in Figure 5.13 by 90 degrees clockwise. This is useful since it allows the assistant to orient the video without the need for the requester to adjust their device.

Switch camera: Sends a message via the data connection that will initiate the logic from Code listing 5.12 on the requester side. This results in the source camera being cycled and the video stream restarted.

Change quality: Sends a message via the data connection that will toggle the video quality on the requester's device.

Take a picture: Sends a message via the data connection that will initiate the picture taking process as documented in Code listing 5.13.

End call: Disposes of the local audio stream and set the *HelperCallStatus* to *callEnded* which will render the page Figure 5.14.

The second section of the assistant user interface will display information streamed from the requester's device. The information includes the accuracy of the GPS service, the

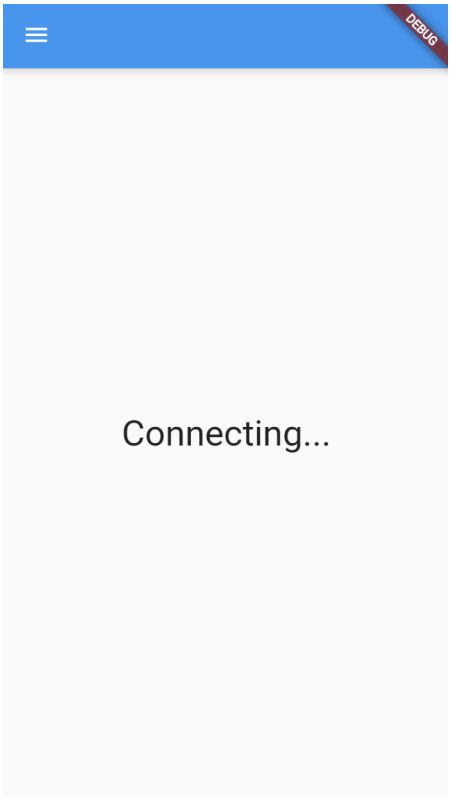

Figure 5.12 Establishing connection page.

speed of the device based on the accelerometer of the device and the location of the device presented via a static Google map embedded element. The map element is obtained from the Google Maps by the use of the requester's latitude and longitude. It is then rendered on the page using an iframe obtained from the webviewx [33] plugin, as detailed in Code listing 5.16.

This approach does not require a Google Maps API key, resulting in no cost for obtaining the map element.

During the video call, the assistant is able to receive pictures from the requester side. These pictures can be viewed by accessing a drawer by clicking on the menu icon on the top left part of the screen, which will result in the page shown in Figure 5.15. Subsequent pictures will be displayed here and stacked in a vertical layout with the most recent picture being placed at the top, this allows the assistant to quickly scroll through the received pictures even when using a mobile device as seen in Figure 5.16. Closing the drawer will make the application render the page in Figure 5.13. Neither the video nor the audio call is interrupted when viewing or receiving pictures.

5.6.4 Call Finished

When *HelperCallStatus* is set to *callEnded*, the view from Figure 5.14 will be rendered. Upon rendering, the resources of *HelperRequester* will be disposed of.

If the user clicks on the *Answer another call* button, the *HelperCallStatus* value will be set to *waitingPartnerId*, resulting in the call cycle being started once again.

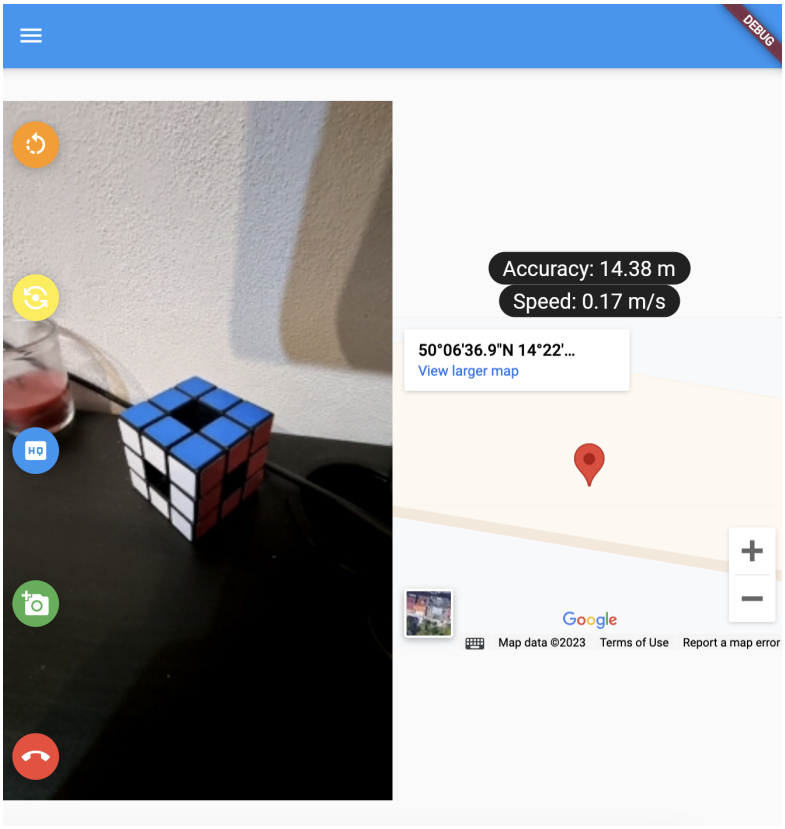

Figure 5.13 Ongoing call on the assistant side.

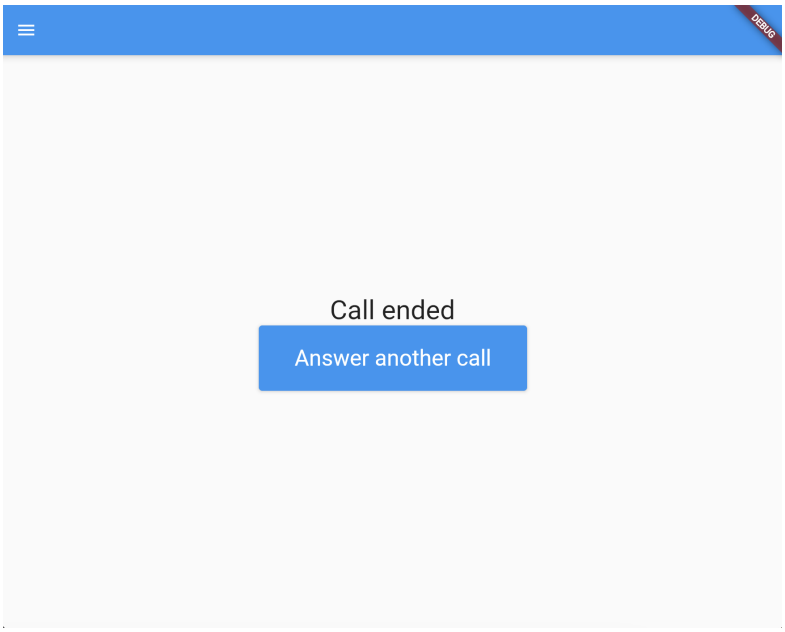

Figure 5.14 Call ended on the assistant side.

```
final String mapQuery =
    'https://maps.google.com/maps? \
    q=$latitude,$longitude&t=&z=20&ie=UTF8&iwloc=&output=embed';

return WebViewX(
  key: const ValueKey('webviewx'),
  initialContent: initialContent,
  initialSourceType: SourceType.html,
  height: screenSize.height / 2.5,
  width: screenSize.width * 0.8,
  onWebViewCreated: (controller) {
    webviewController = controller;
    webviewController.loadContent(mapQuery, SourceType.url);
  },
  mobileSpecificParams: const MobileSpecificParams(
    androidEnableHybridComposition: true,
  ),
  navigationDelegate: (navigation) {
    debugPrint(navigation.content.sourceType.toString());
    return NavigationDecision.navigate;
  },
);
```

Code listing 5.16 Code for rendering an embedded map in Flutter.

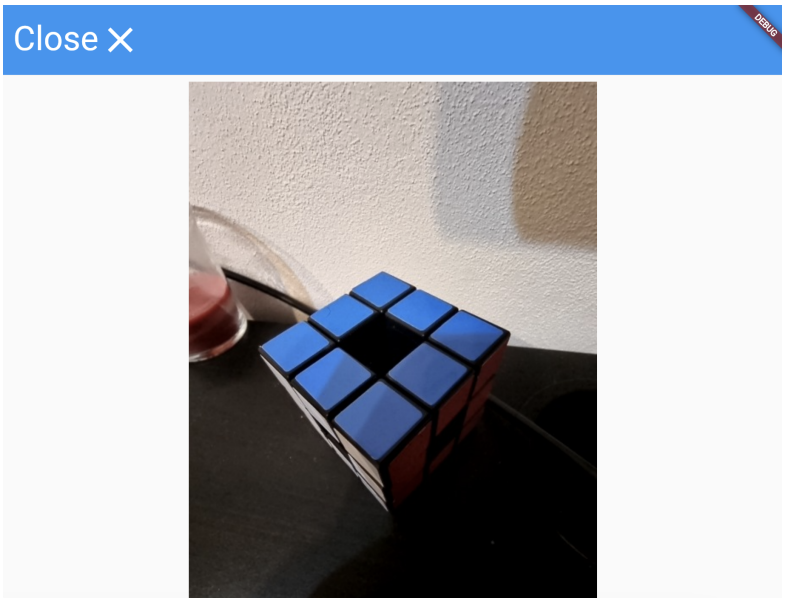

Figure 5.15 Drawer for received images, viewed on a desktop computer.

5.7 Message Transmission

Certain events on both flows generate messages that need to be transmitted between both parties. In order to ensure the transmission of the messages, a *DataConnection* object exists on both sides. Messages are classified in *PeerMessageType* and *ActionMessageType*.

Figure 5.16 Drawer for received images with multiple pictures on a mobile device.

The different types of messages are shown in Code listing 5.17.

Messages sent through a *DataConnection* object need to be in the form of a *Map* with String keys and values. If a value is of a different type, such as an object, number or boolean, an error will occur when transmitting the message.

A *geoLocation* message will be generated using the lines from Code listing 5.18. The *position* object will be obtained from a callback generated by the geolocator plugin [29].

A *speed* message is generated using the lines from Code listing 5.19. The *UserAccelerometerEvent* object is obtained from the sensors_plus plugin [30], which contains absolute values corresponding to the current speed of the device in the x, y and z axis.

A *picture* message will be generated using the lines from Code listing 5.20. The *base64Image* is a String representing an image taken by the requester's device.

An *action* message will be composed of a dictionary containing a single key-value pair, as listed in Code listing 5.21. This type of message will only originate from the assistant and will be sent to the requester.

Once the requester receives an action message, it will be decoded and an action will be taken, depending on the *data* value with each possible value corresponding to one of the four actions available to the requester.

5.8 TURN Server

During development, the application had problems initiating the video call under certain conditions such as when using an Android device with mobile data and a VPN service. This problem was temporarily solved by the configuration of a TURN (Traversal Using Relays around NAT) server. The TURN server will be used to relay information between devices in case that the network used masks the IP address and the information provided

```
enum PeerMessageType {
    geoLocation,
    speed,
    picture,
    action
}

enum ActionMessageType {
    changeCamera,
    changeQuality,
    takePicture,
    endCall
}
```

Code listing 5.17 Enums representing the message types.

```
Map<String, String> getEncodableLocationMessage(Position
  position) {
    return {
        typeKey: PeerMessageType.geoLocation.name,
        "latitude": position.latitude.toString(),
        "longitude": position.longitude.toString(),
        "accuracy": position.accuracy.toString()
    };
}
```

Code listing 5.18 Class for a geolocation message.

```
Map<String, String> getEncodableAccelerationMessage(
  UserAccelerometerEvent event) {
    return {
        typeKey: PeerMessageType.speed.name,
        "x": event.x.toString(),
        "y": event.y.toString(),
        "z": event.z.toString(),
    };
}
```

Code listing 5.19 Class for a speed information message.

```
Map<String, String> getEncodablePictureMessage(String
  base64Image) {
    return {
        typeKey: PeerMessageType.picture.name,
        "data": base64Image,
    };
}
```

Code listing 5.20 Class for a picture message.

```
Map<String, String> getEncodableActionMessage(ActionMessageType
  action) {
    return {typeKey: PeerMessageType.action.name, "data":
      action.name};
}
```

Code listing 5.21 Class for an action message.

is not sufficient to start a peer-to-peer session [34]. This, of course, has the drawback of added latency to the call and incurring extra costs to run the server. For this reason, a free WebRTC TURN server was configured using Open Relay, a TURN service that provides enterprise-grade reliability and support For both TCP and UDP connections [35]. This addition fixed the communication problems for certain edge cases but it would disable all peer-to-peer capabilities when accessing the application via a Safari web browser on an iOS device. For this reason, this configuration was discarded during development.

5.9 Evaluation of the Implementation

After describing the implementation of the application we can synthesize how the problems discussed in Section 3.1 have been addressed by the solution:

1. The maximum number of interactable elements present at a single moment on the requester side has been limited to four, which allows the accessibility features to cycle through them efficiently.

2. The selection of a lower resolution for the outgoing video stream on the requester side provides some control over the data consumption while using the application.

3. The features needed for video and audio streaming, picture sending and location sharing have been included in a single application preventing the need for the user to switch between multiple applications when receiving remote help.

4. During the development, the technologies that provided official support for mobile and web platforms were selected, this includes the framework and Flutter libraries. Extensive testing was made with the use of both emulated and physical devices. This ensured that support for the offered features across multiple platforms has been achieved.

5. The low number of displayed elements on the requester side combined with the responsive nature of the framework ensures that elements will not overlap or become inaccessible on most mobile devices.

6. The flow of the application on the requester side was designed to be sequential and does not include timed action to which the user has to react swiftly.

7. The only text input on the requester side is located on the settings page (Figure 5.5). The value of this input is persisted locally, so the user does not spend an extended amount of time typing into the application. The support for quickly sharing the call id, as shown in Figure 5.8, also saves time since the user does not need to type the name of the person they wish to share the code with or the name of the application they want to share it with.

8. The option to select the preferred camera for video streaming is provided and is persisted. In the case of selecting an audio source, this is an option that can be set globally for the web browser via accessibility options.

9. The GPS location is shared as well as its accuracy and the current speed of the device. This provides the assistant with more tools to properly aid the requester in navigation.

5.10 Deployment

The current implementation is able to be compiled into Android and iOS application packages as well as a web application. Currently, the only publicly available version of the application is the web version. This version is hosted using GitHub pages [1], a static web hosting service that serves the content of a website which can be updated by modifying the source which will trigger a Jekyll pipeline and update the web page [36]. This service is free with the limitations of repository sizes of up to 1 GB, a bandwidth limit of 100 GB per month after which the speed will be throttled, a build limit of 10 build events per hour and rate limiting [37]. This is enough to cover the usage for testing and production.

[1] `https://hidalgoivan.github.io/#/`

Chapter 6

Feedback and Interviews

How do different users with visual impairments navigate the world and what insights can we gather from their unique experiences? What feedback do they have for existing solutions and what would their ideal application offer? In this chapter, we will write about personal experiences from visually impaired users and the feedback they have for the accomplished work.

During the development of this project, multiple meetings and interviews were conducted with fully blind and partially sighted users in SONS. The following text is a chronological synthesis of insights and feedback we received from them.

6.1 Meeting #1, January 16th, 2023

For the first meeting, we prepared exploratory questions in order to collect information about which applications are used most frequently and what changes they would appreciate having for them. For each response, a condensed version combining the feedback from all attendants is provided.

Which applications do you use on a day-to-day basis?

Day-to-day activities are aided by a variety of applications. Some of them are Seznam Mapy, a mobile application for maps and navigation that supports downloading regional maps for use without an internet connection [38]. Zuzanka, an iOS-only application that is able to read the expiration date of physical products by scanning them with the camera of the device and provides such features as saving the expiration date on a calendar application and guidance for common places where expiry dates are printed based on a cumulative database of expiration date pictures [39]. OKO, an iOS and Android application designed to help partially sighted users navigate street crossings by using AI to detect the color of a traffic light using the camera of the device and producing an auditory queue to alert the user about the state of the traffic light. Google Maps is frequently used to check if the desired location has been reached and for navigating the city with the description provided by the application while it is in pedestrian mode.

What would you change about these applications?

The user experience is very inefficient for some applications since it requires multiple

inputs from the user to get the desired task done. This problem is compounded when the user is using some accessibility feature, which results in a simple task taking a relatively long amount of time to complete and it becomes even worse when the user is stressed or needs to act quickly. A navigation application designed for pedestrian use is sorely needed since most current applications support a pedestrian mode but do not provide the same features as in the mode used for vehicles. Certain applications provide a navigation description for getting from one point to another, but this is frequently not in enough detail to navigate an urban setting efficiently. Since most cities are not organized in grids, the complexity of the movements needed is not expressed appropriately. OKO will sometimes incorrectly announce the color of traffic lights or will pick up a different traffic light which is also in view of the camera, which can lead to very dangerous situations when crossing streets. This inaccuracy is something that would really be appreciated to be corrected. For most navigation applications, GPS accuracy is a big hindrance, since in urban environments, the precision of location services can be very low. It limits the usability of such applications.

What would your perfect application be like?

A perfect application for assisting visually impaired users would read the house numbers in real-time when traversing the street, promptly notify users of errors in route navigation such as taking the wrong route or waiting on the wrong side of the street for public transport, communicate to public transport conductors that a visually impaired person is waiting on the next stop so they may provide assistance, and be able to read text on any type of display since modern devices have problems detecting numbers, for example on washing machine displays. A solution involving augmented reality lenses would be really useful if it could detect obstacles on the way and sort them by importance. This would function similarly to a white cane, which can only provide tactile feedback about obstacles but not identify them or take affirmative action to inform the user about danger or if an action from their side is required.

6.2 Meeting #2, February 1st 2023

For this meeting, we asked users with different visual impairments to send a text message or search for the weather in a web browser on their mobile devices. The following is a condensed report of the observations we were able to capture and the feedback provided by the users.

Voice-over functionality can be enabled by performing a gesture on iPhone devices or it can also be configured to start by default. After an update, the voice quality of the screen reader decreased notably, which negatively impacted the usability of this feature. The voice-over functionality sometimes replaces certain character combinations by different words, which is especially frequent when reading abbreviations, and seriously hinders usability. It does not announce when it interprets the text in this way and it takes time to learn which abbreviations or combinations are picked up by this feature.

Some applications display a menu with quick actions when interacting with voice-over enabled. This allows the user to skip large portions of the user flow to get tasks done quicker. An example of this is the ability to write emails from this menu without opening the email application. Sorting the icons of the applications helps users quickly access the application that they want to use. For this reason, they frequently have multiple folders with similar applications inside. For starting an application, some users prefer to use

voice dictation instead of tapping on the application icon. This is not very reliable, since the voice dictation functionality is frequently only supported for English, which makes applications that are named in a different language such as Czech unrecognizable by the device by name.

Memorization is a large part of the application usage. After a user has gone through some task enough times, they can memorize where the location of the important elements are, and in future uses, they can quickly tap directly on such elements instead of scrolling through the available user interface components with accessibility options. When reading text, a bold font is preferred by partially sighted users since it is easier to read when using features such as zoom on the screen and using extra tools such as magnifying glasses. Setting the font size is inconsistent since some applications ignore the default size imposed by the system. When inputting text, different methods are used. A straightforward strategy is the use of a Bluetooth keyboard with Braille keys. This allows the user to quickly input large amounts of text and operate the device with efficiency with the drawback of needing to carry the keyboard around and charging it from time to time. When using the screen keyboard provided by the device, the voice-over will read every letter when the user hovers over it. This results in a very inefficient user experience, since some letters sound very similar. The solution provided is that the voice-over will use the selected letter in a word such as 'Emil' when the user hovers over the letter 'E'. This word pronunciation takes a lot of time to complete and renders this strategy relatively slow and difficult to follow. An on-screen Braille keyboard is also supported, for which the user holds the phone horizontally with both thumbs on the back side of the device and the other fingers are used to tap on the edges of the screen, corresponding to a single dot in Braille notation. This method takes some training and time to learnbut is a very viable alternative for inputting large quantities of text.

After this demonstration, we presented our ideas for the prototype that would later become the solution described in this thesis. After hearing about our proposal, the following features were requested: adding the option to adjust the quality of the video during the call, sharing the GPS location of the user with estimated accuracy, ability to send a picture from the requester side in high definition, allow the assistant to adjust the quality of the video stream from their device, and also start the process for taking a picture without any input from the requester.

6.3 Meeting #3, March 3rd 2023

Before this meeting, the functional requirements were defined and different technologies were tested for the development of the application. For this meeting, the fourth iteration of the prototype was presented and a video and audio call was achieved with the requester side using an Android phone and the assistant side using a computer running Windows and using Google Chrome as a web browser. The feedback from this meeting was positive and the ability of the application to function via a web page accessible by the hardware in the navigation center at SONS was a very welcomed aspect of this project.

6.4 Successive Meetings March 22nd – April 12th 2023

During the following meetings, multiple iterations of the final solution were presented and tested on multiple devices. Several bugs and faulty features were identified and corrected

thanks to the testing performed. Here is a condensed list of important observations made during these meetings:

- When starting a call, the phone would sometimes switch the audio input device. This can make it difficult for requester users to communicate since they frequently make use of headphones with integrated microphones.
- The *SelectableText* widget provides the option for the user to select and copy text displayed on the screen [40]. This widget is described as a disabled text input on certain mobile web browsers, which can be confusing for a user going to the call flow for the first time.
- The use of a VPN service can interfere with the peer-to-peer connection, preventing the requester and the assistant user to establish a communication channel.
- Some users may experience difficulty connecting to the application due to firewall restrictions placed on the network they are currently using; extra testing is necessary to come up with solutions to this problem.
- The navigation center at SONS is very much pleased with the achieved progress and expects to start using the application to help visually impaired people in the near future.
- When comparing the audio delay in this solution to the delay present in dedicated audio call options, this solution frequently has a lower latency resulting in reduced audio transmission delay.

Chapter 7
Conclusion

What can we learn from this work? How did we manage the input from volunteers from SONS and what is the future of the project moving forward?

The conducted research has given a comprehensive overview of the current state of solutions available for users who are fully blind or partially sighted. There are multiple ways visually impaired users operate their mobile devices and technological advances have greatly improved the tools they can make use of to help them become the most independent, creative and self-reliant version of themselves possible. While these tools allow partially sighted users to operate most of applications available for mobile devices and navigate many websites, the current implementation is frequently not sufficient to guarantee a usable experience or even expose all of the features a solution may offer.

Through interviews, observations and experiences gathered from cumulative years of living with a sight condition, we have been able to identify frequently requested features and changes that would greatly improve the ability of visually impaired users to navigate the world and help developers better understand their needs and make changes that can have a major positive impact on the experience of their visually impaired users. In close cooperation with the SONS members, we have designed an easy-to-use, free and reliable application that is being used by the navigation centre at SONS to aid in providing support to users who need it. Frequent complaints about current solutions have been addressed and kept in mind when implementing the features for the final solution. After much experimentation, research and prototype iterations, the final solution has been deployed and has been accepted by the target audience.

The list of problems in Chapter 3 has been addressed by limiting the maximum number of interactable elements, providing the option for a lower resolution of the video stream, including multiple requested features into a single application and selecting a technology that is able to be deployed to multiple platforms, among other strategies.

The implementation does not outright solve all of the problems in their entirety, but it does provide some level of support that attempts to minimize the impact of the problems on the user experience. One example is the GPS coordinates: while the application cannot always provide a perfect GPS location, it at least provides extra information that allows the assistant to form a better idea of the real location of the requester. Each one of these problems has been kept in mind when developing the application and the solutions have been reached via a combination of technical research and brainstorming with users at SONS.

This application covers the requested functionality in at least a usable manner; however, there are still several limitations. Selecting a camera for taking pictures is not supported. Testing is a lengthy process since inconsistencies between emulators and real devices have been found multiple times during the development process. Implementation of new features is a particularly tedious process since although the added plugins officially support multiple platforms, in practice, the behavior of certain plugins is inconsistent and sometimes problems with the technologies arise for which there is very little to no documentation. The application is heavily dependent on the public PeerJS server, which means that if the server is down or there are too many concurrent users trying to make a connection, the performance of the application could be negatively impacted. Accessing and sharing the orientation of the device's compass is currently not supported. Certain mobile web browsers will not allow the editing of the text input on the settings page.

7.1 Future Work

The current implementation is usable for the visually impaired users and the navigation center, however the meetings and continued collaboration with SONS has provided a list of features that would be desirable to have in future implementations. Some of these observations have been labeled as future work:

- **Deploying a private peer-to-peer server:** The application currently uses the public server provided by the PeerJS library [16] to initiate the connection between devices. This is serviceable for the purpose of the application. However SONS has access to a virtual server which is capable to run its own version of the server, by doing this more control could be provided for the availability of the application and generation of server-side peer ids.
- **Publishing mobile application:** The application has been successfully compiled to both Android and iOS packages. Publishing to Google Play and App Store would significantly increase the reach of the solution and would fix problems that are unique to the web version of the application.
- **Adding extra settings:** The addition of extra settings such as selecting the pitch, speed and volume of the text-to-speech voice, selecting whether to adjust the font size to that of the system default or having a custom size and being able to toggle location sharing.
- **Extra assistant controls:** The ability to let the assistant party switch the layout to present video only, location only or received pictures only.
- **Proactive calls:** The ability to actively call a contact without having the second party open the application and start a session.
- **Call recovery:** Automatically try to recover a call in case the connection is briefly lost or there has been a network change.
- **Received image controls:** Adding controls to the received images on the assistant side. These controls would allow the assistant to resize, rotate and zoom into specific sections of the pictures.

Bibliography

1. *Country Map & Estimates of Vision Loss - Czech Republic* [The International Agency for the Prevention of Blindness (IAPB)]. Accessed on 2023-04-18. Available also from: `https://www.iapb.org/learn/vision-atlas/magnitude-and-projections/countries/czech-republic/`.
2. *Refractive Errors* [National Eye Institute]. Accessed on 2023-04-18. Available also from: `https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/refractive-errors`.
3. *Causes of Vision Loss* [The International Agency for the Prevention of Blindness (IAPB)]. Accessed on 2023-04-18. Available also from: `https://www.iapb.org/learn/vision-atlas/causes-of-vision-loss/`.
4. *Cataracts* [National Eye Institute]. Accessed on 2023-04-18. Available also from: `https://www.nei.nih.gov/learn-about-eye-health/eye-conditions-and-diseases/cataracts`.
5. *About us* [SONS ČR]. Accessed on 2023-03-03. Available also from: `https://www.metered.ca/tools/openrelay/`.
6. *Android accessibility overview* [Google]. Accessed on 2023-02-01. Available also from: `https://support.google.com/accessibility/android/answer/6006564?hl=en&ref_topic=6007234`.
7. *Advanced Braille Keyboard Tutorial Part 4 : Different types of Braille Screen Input* [Nalin]. Accessed on 2023-05-03. Available also from: `https://www.youtube.com/watch?v=EPRdsR78mOk`.
8. *About the vision accessibility features on your iPhone or iPad* [Apple]. Accessed on 2023-02-01. Available also from: `https://support.apple.com/en-us/HT210076`.
9. *About section* [Be My Eyes]. Accessed on 2023-02-06. Available also from: `https://www.bemyeyes.com/about`.
10. *Be My Eyes* [Be My Eyes]. Accessed on 2023-05-03. Available also from: `https://play.google.com/store/apps/details?id=com.bemyeyes.bemyeyes`.
11. *Ionic Framework* [Ionic]. Accessed on 2023-02-09. Available also from: `https://ionicframework.com/`.
12. *We help companies build world-class apps.* [Ionic]. Accessed on 2023-04-21. Available also from: `https://ionic.io/customers`.

13. ANCHETA, Wern. *React Native Web vs. Flutter web*. Accessed on 2023-03-14. Available also from: https://blog.logrocket.com/react-native-web-vs-flutter-web/.

14. *Apache Cordova, overview* [Cordova]. Accessed on 2023-03-14. Available also from: https://cordova.apache.org/docs/en/latest/guide/overview/index.html.

15. *The Good and the Bad of Ionic Mobile Development.* [AltexSoft]. Accessed on 2023-04-21. Available also from: https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-ionic-mobile-development/.

16. *The PeerJS library* [PeerJS]. Accessed on 2023-02-28. Available also from: https://peerjs.com/.

17. MINTO, Rob. *The genius behind Google's browser*. Accessed on 2023-04-22. Available also from: https://web.archive.org/web/20111201001419/http://www.ft.com:80/cms/s/2/03775904-177c-11de-8c9d-0000779fd2ac.html#axzz1fEmNkUnB.

18. *The Dart type system* [Dart]. Accessed on 2023-04-22. Available also from: https://dart.dev/language/type-system.

19. BRACHA, Gilad; BARK, Lars. *Presentation: "Opening Keynote: Dart, a new programming language for structured web programming"*. Accessed on 2023-04-23. Available also from: http://gotocon.com/aarhus-2011/presentation/Opening%20Keynote:%20Dart,%20a%20new%20programming%20language%20for%20structured%20web%20programming.

20. BAK, Lars; LUND, Kasper. *Dart for the Entire Web*. Accessed on 2023-04-22. Available also from: https://news.dartlang.org/2015/03/dart-for-entire-web.html.

21. *TC52 - Dart* [European Computer Manufacturers Association]. Accessed on 2023-04-22. Available also from: https://web.archive.org/web/20160802100651/http://www.ecma-international.org/memento/TC52.htm.

22. *FAQ* [Flutter]. Accessed on 2023-04-22. Available also from: https://docs.flutter.dev/resources/faq.

23. *Sky: An Experiment Writing Dart for Mobile* [Google Developers]. Accessed on 2023-04-22. Available also from: https://www.youtube.com/watch?v=PnIWl33YMwA.

24. *Supported deployment platforms* [Flutter]. Accessed on 2023-04-22. Available also from: https://docs.flutter.dev/reference/supported-platforms.

25. KARMACHARYA, Samriddhi. *LifeCycle Methods Of Flutter Widgets*. Accessed on 2023-04-22. Available also from: https://flutterguide.com/lifecycle-methods-of-flutter-widgets/.

26. KAPLAN, Muhammed. *peerdart 0.5.0*. Accessed on 2023-03-01. Available also from: https://pub.dev/packages/peerdart.

27. *peerjs-server* [PeerJS]. Accessed on 2023-02-09. Available also from: https://github.com/peers/peerjs-server.

28. LESNITSKY, Andrei. *localstorage 4.0.1+2*. Accessed on 2023-04-09. Available also from: https://pub.dev/packages/localstorage.

29. *geolocator 9.0.2* [Baseflow]. Accessed on 2023-03-01. Available also from: https://pub.dev/packages/geolocator.

30. *sensors_plus 2.0.2* [Flutter community]. Accessed on 2023-03-18. Available also from: https://pub.dev/packages/sensors_plus.

31. *Share plugin 6.3.1* [Flutter community]. Accessed on 2023-03-01. Available also from: https://pub.dev/packages/share_plus.

32. *camera 0.10.3+2* [Flutter.dev]. Accessed on 2023-03-20. Available also from: https://pub.dev/packages/camera.

33. FLUTUR, Adrian. *webviewx 0.2.2*. Accessed on 2023-03-02. Available also from: https://pub.dev/packages/webviewx.

34. *What are STUN and TURN Servers? (WebRTC Tips from WebRTC.ventures)* [WebRTC.ventures]. Accessed on 2023-03-06. Available also from: https://www.youtube.com/watch?v=4dLJmZOcWFc.

35. *Open Relay: Free WebRTC TURN Server* [Next Path Software Consulting Inc.]. Accessed on 2023-03-03. Available also from: https://www.metered.ca/tools/openrelay/.

36. CLARK, Barry. *Build A Blog With Jekyll And GitHub Page*. Accessed on 2023-04-30. Available also from: https://www.smashingmagazine.com/2014/08/build-blog-jekyll-github-pages/.

37. *About GitHub Pages* [GitHub]. Accessed on 2023-04-30. Available also from: https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages#basics-of-github-pages.

38. *Mapy.cz navigation & maps* [Seznam.cz a.s.]. Accessed on 2023-04-28. Available also from: https://apps.apple.com/us/app/mapy-cz-navigation-maps/id411411020.

39. *Zuzanka* [Zatoichi Sp. z o.o.]. Accessed on 2023-04-28. Available also from: https://apps.apple.com/az/app/zuzanka/id6444279925.

40. *SelectableText class* [Flutter]. Accessed on 2023-03-01. Available also from: https://api.flutter.dev/flutter/material/SelectableText-class.html.