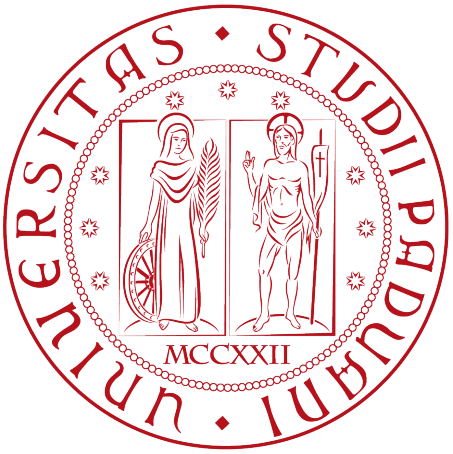
University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Degree in Computer Science



NetCare: a cross-platform mobile application to ensure the quality of life of caregivers having patients with chronic diseases

Master thesis

Supervisor
Professor Ombretta Gaggi
Co-supervisor
Professor Sabrina Cipolletta

Graduate student
Filippo Brugnolaro
Student id 2087666

Academic Year 2023-2024

Abstract

This research project aims to develop a proof of concept of a digital system (*NetCare*) which supports family caregivers of people with cognitive decline and connects them with each other. The main goal is to offer an effective tool to improve the caring role, but also to promote family caregivers' well-being and fight against the risk of loneliness and social isolation. Literature about this topic is growing, but it is lacking in long-terms experiments.

NetCare offers a platform where caregivers can create groups in which they can include both patients and other caregivers. Thanks to these groups, caregivers can create, edit and share events, assign activities to patients and receive updates on their completion. Users can also jot down notes and share them with other group members. The system supports multimedia for notes, events and activities, enhancing the overall user experience. In addition, caregivers also have access to a wide range of tips focused on their needs and their loved ones' health, but always focusing on caregivers' well-being.

Thanks to a highly interdisciplinary approach that combines medical, psychological, computer science and information technology knowledge, the project focuses on the creation of a connection between the scientific community and healthcare sector.

"Good programmers know what to write. Great ones know what to rewrite and reuse"

— Eric Steven Raymond

Acknowledgements

First of all, I would like to express my gratitude to the supervisor of my thesis, Professor Ombretta Gaggi, for her availability and help during the thesis' drafting process.

I would like to express my heartfelt gratitude to my family for their unwavering support and closeness given all the time. Thank you for never letting me lack anything during my years of study.

I would like to thank my friends who have been close and supported me over these years, especially during difficult moments.

Finally, a special thank you goes to my friends Alessandro and Francesco, who made studying less burdensome and shared many enjoyable moments with me.

Padova, September 2024 Filippo Brugnolaro

Contents

List of Figures

List of Tables

List of source codes

Chapter 1

Introduction

Nowadays, the world population is registering an increase in average life expectancy and people are used to live over 75 years in a lot of countries with projections rising every year (see Figure 1.1) [15] [36] [41]. On the same way, the global health caregiving market was valued $210.53 billion in 2024 and projections say it will reach nearly $377.69 billion by 2029, with a rise in *Compound Annual Growth Rate* of 12.4% from 2024 to 2029 [71]. For these reasons, it is clear that elderly people's care is very important and has a huge impact in our society and in daily life of people.

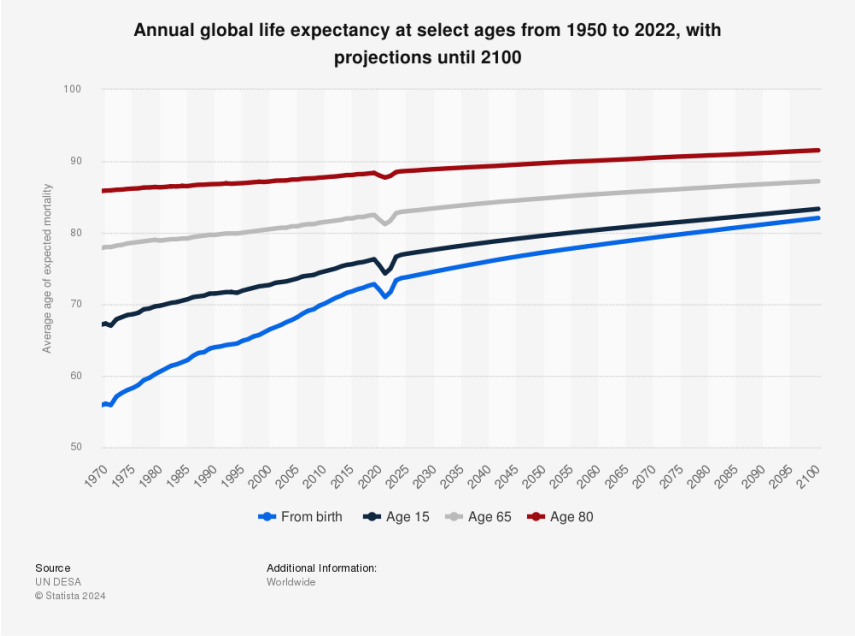


Figure 1.1: Annual global life expectancy from 1970 to 2022 with projections until 2100 [36]

There is a specific category of caring which involve the people with mild and major cognitive decline (i.e., dementia) which is substantially relying on the so-called *informal caregivers*, who are usually family members like spouses, children and sometimes both of them. Figure 1.2 shows that the majority of informal caregivers are middle-aged adults (73% are women). To fulfill their caregiving responsibilities, they are often

forced to make drastic changes into their life, such as reducing their work schedule and their social life, having dramatic consequences on the local economy and on their psychological well-being. In fact, a report by the *Alzheimer Association* [32] revealed that only in U.S. the unpaid care value is estimated in 18.4 billion hours, which are roughly $346.6 billion, and the direct costs for families are about $360 billion. In the same study, a comparison between different types of caregiver showed that the amount of work generated for the people living with Alzheimer's or other dementias is significantly higher rather than the ones living with other older people as shown in Figure 1.3. This implies also that nearly 60% of Alzheimer's and dementia caregivers rate the emotional stress of caregiving as high or very high and 40% of family caregivers of people with Alzheimer's and other dementias suffer from depression. Finally nearly 75% of Alzheimer's and dementia caregivers are somewhat or very concerned about maintaining their own health since becoming a caregiver [32].

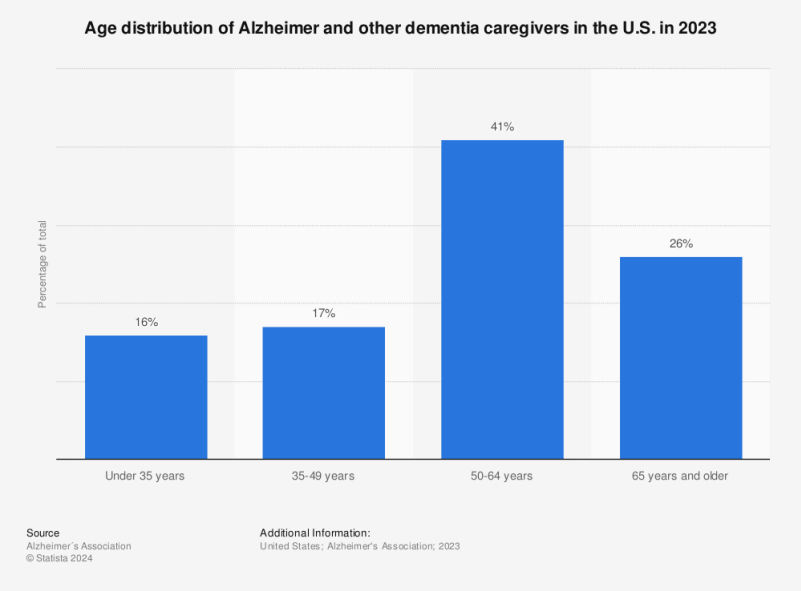


Figure 1.2: Age distribution of Alzheimer and other dementia caregivers in U.S. in 2023 [32]

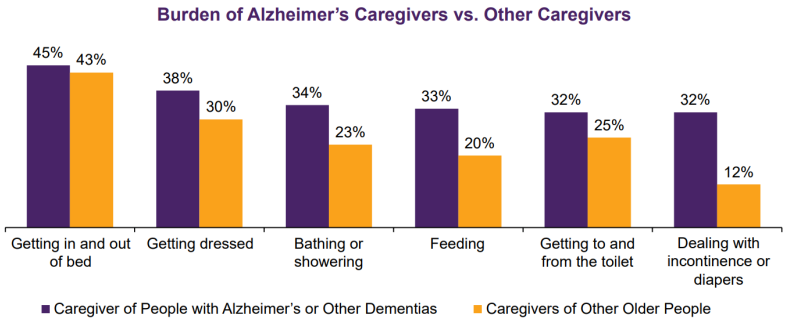


Figure 1.3: Burden of Alzheimer's Caregivers vs. Other Caregivers [32]

So one of the major difficulties experienced by informal caregivers is the perceived sense of emotional and social loneliness, that is the felt lack of attachment to a significant other, and of a social network or community [25]. Being associated with several consequences on both mental and physical well-being, loneliness has been recently defined as a key public health concern [18] [21], which needs to be tackled with effective interventions.

In response to these pressing issues, various initiatives and programs have been developed to provide support for informal caregivers. However, despite these efforts, the gap in adequate support systems remains significant (as will be discussed in Chapter 2) and continued innovation and investments in caregiver support are necessary to address the growing demand.

1.1 Contribution

In this work we tried to fill the gaps of existent applications by defining and intuitive and fancy design. We took into consideration which is the typical end user and adapted the user experience for the target. In particular, we created a cross-platform mobile application and the main goal is trying to reduce the burden generated by the work that caregivers have to do. By focusing on user-friendly features and a visually appealing interface, we attempted to create a tool that enhances efficiency and ease of use for caregivers in their daily tasks.

The significance of this work is related also to the caregivers crucial role in society, who often work very hard to ensure the well-being of those they care for. Our application aims to directly address these issues by providing a reliable and efficient tool that simplifies and optimize the organization of their daily responsibilities.

Moreover, by reducing the burden on caregivers, we contribute to improving the overall quality of care provided to patients and loved ones. When caregivers have access to tools that enhance their efficiency and reduce their stress, they can dedicate more time and energy to other social activities which are equally important for their well-being. On the other side, this can lead to better outcomes for both patients and caregivers.

Additionally, the cross-platform nature of our application ensures that it is accessible to a wide range of users, regardless of the type of device they prefer. This inclusivity is important in order to reach as many caregivers as possible, providing them with the support they need without considering any of their technological preferences and guaranteeing a consistent user experience from any device.

Our emphasis on an intuitive and visually appealing design is not only aesthetic; it is about creating an environment where users feel comfortable and confident in navigating the application. An intuitive interface reduces the learning curve and allows caregivers to quickly and easily access the features they need, minimizing frustration and maximizing productivity.

In summary, this work is important in providing all the needed support to caregivers, enhancing their efficiency, reducing their stress and improving the quality of care provided. Our goal is not only to improve the tools already available, but to create a more supportive environment for caregivers, ensuring they have the resources they need to provide the best possible care with an eye on their health which is equally important with respect to the patients ones. By addressing the gaps in existing applications with a user-centric approach, we hope to make a meaningful and lasting impact on the lives of both patients and caregivers.

1.2 Outline

In this section, we describe the next chapters' organization:

- **Chapter 2:** it reviews current applications designed for caregivers, identifying their gaps and limitations. It also summarizes research studies related to caregiver support tools, analyzing findings that could be useful for the design and functionalities of the new application;
- **Chapter 3:** it explains the primary goals of the application, detailing the specific problems the app aims to solve for caregivers from the review of the literature. This chapter describes the typical end users, and lists the core features of the application, explaining how these address the needs of caregivers;
- **Chapter 4:** it justifies and provides an overview of the development frameworks, tools and services used;
- **Chapter 5:** it details the design process of the application, describing and motivating the application's architecture and libraries used. It is treated also the topic about mobile design analysis and accessibility. Finally, it is described the testing code phase and the framework defined for the evaluation phase of the application in real-world scenarios, including feedback from psychologists based on their field experience;
- **Chapter 6:** it recaps the project's achievements and discusses the anticipated impact of the application on caregivers' daily tasks. It identifies any limitations encountered during the project and suggests potential features and improvements for future versions of the application.

Chapter 2

Related works

2.1 Introduction

In recent years, several technological interventions have been created to support caregivers of patients with dementia. They differ in duration and type, but primarily offer information on caregiving and dementia, such as web-sourced information, practical advices, quizzes and forums (e.g., A Walk Through Dementia [28], Dementia Talk App [55]). Social support were provided through online groups and networks that connect caregivers and facilitate the sharing of patient information. (e.g., Inlife [8] and Carely [49]). We review a lot of papers to identify the current findings and determine any valuable information useful to decide which are the features that are more appreciated by patients and caregivers.

2.2 Methodology used for searching papers

We outline the methodology used for searching and selecting relevant papers to gain a better understanding of the current state of the art on the topic of caregivers and mobile applications.

2.2.1 Search Strategy

In order to identify and review literature on solutions to improve the well-being among caregivers, especially focusing on mobile applications, a comprehensive search strategy was developed. This strategy was used in order to select a wide range of studies trying to exclude the less relevant ones. Particular attention was paid to studies that explored the design, the needs and the effectiveness of mobile applications as well-being improvement tools for caregivers.

2.2.2 Databases and Search Engines

The search was conducted using academic databases and search engines to create a first wide collection of studies. These ones are presented in order of usage:

- Google Scholar;
- PubMed;

- ResearchGate.

2.2.3 Keywords and Search Terms

A combination of keywords was used to maximize the retrieval of pertinent studies. These keywords were selected based on the focus of the research and included terms related to caregivers, stress and mobile applications. The primary search terms included:

- Caregiver stress management;
- Help informal caregivers;
- Mobile apps caregivers;
- Applications for caregiver review;
- Alzheimer's caregivers stress;
- Dementia support caregivers;
- Applications for caregivers.

2.2.4 Inclusion and Exclusion Criteria

To ensure the relevance and quality of the selected studies, specific inclusion and exclusion criteria were established. Sometimes, if one or more criteria were not fully met by the work, but it contained useful information, it was therefore included as a paper of secondary relevance.

Inclusion Criteria:

To identify relevant, high-quality studies, the following inclusion criteria were established:

- **Focus on mobile applications:**
  - studies have to focus specifically on mobile applications or digital platforms that are designed to support caregivers;
  - the interventions can include apps for stress management, health monitoring, educational resources, communication tools and other digital aids that assist caregivers in their role.
- **Language:** only papers written in English are included to ensure that the research is accessible and comprehensible to the reviewers;
- **Publication date:** works have to be conducted and published within the last five years. This criterion ensures that the findings are current and relevant to the present technological advancements and caregiving practices;
- **Impact on caregivers:** studies have to address the impact of digital tools on caregivers. This includes, but is not limited to, improvements in caregiver well-being, reductions in caregiver stress, enhanced caregiving skills.

Exclusion Criteria:

To eliminate irrelevant, low-quality studies, the following exclusion criteria were established:

- **Irrelevance to caregivers:** studies that are not specifically referred to caregivers or do not address caregiver stress and related issues are excluded. This ensures that all selected studies are pertinent to the primary focus of the review;
- **Focus on professional caregivers:** researches that only focuses on professional caregivers, such as nurses, home health aides or other healthcare professionals, are excluded. The primary interest is on informal caregivers who provide unpaid care to family members;
- **Lack of empirical data:** papers that do not provide empirical data or lack a proper evaluation of the interventions are excluded. This includes theoretical papers, opinion pieces and those that do not present measurable outcomes or data-driven insights.

Secondary Relevance:

In some instances, studies that do not fully meet all the primary inclusion criteria, but contain useful information, may be included as papers of secondary relevance. These studies might offer valuable context, supplementary data or insights that contribute to a broader understanding of the topic.

2.2.5 Selection Process

The initial search included a large number of studies. The selection process involved several steps to exclude the papers maintaining only the most relevant ones:

- **Title and abstract screening:** titles and abstracts of the retrieved papers were screened to identify studies that potentially met the inclusion criteria. Studies that did not clearly relate to caregiver burden reduction or mobile applications for caregivers were excluded at this stage;
- **Full-text review:** the full text of the remaining papers was reviewed in detail. This step ensured that the studies indeed met all the inclusion criteria and provided sufficient data on the impact of mobile applications;
- **Quality assessment:** selected studies were assessed for work quality using established criteria such as the presence of a clear hypothesis, description of the intervention, study design, data collection methods and analysis techniques.

Additionally, data from the selected studies were systematically extracted and synthesized to provide a comprehensive overview of the current topic. This included identifying key outcomes related to the topic, evaluating the effectiveness of different mobile application features and highlighting any gaps in the current research. The synthesis aimed to have meaningful conclusions about the overall impact of mobile applications on caregivers. This rigorous selection and analysis process ensured that the final review was both comprehensive and reliable, providing valuable insights for leveraging mobile technology to support caregivers. Figure 2.1 represents the previously mentioned steps. The diagram includes the number of papers at each stage, allowing to track the reduction in the number of papers as the review progresses.

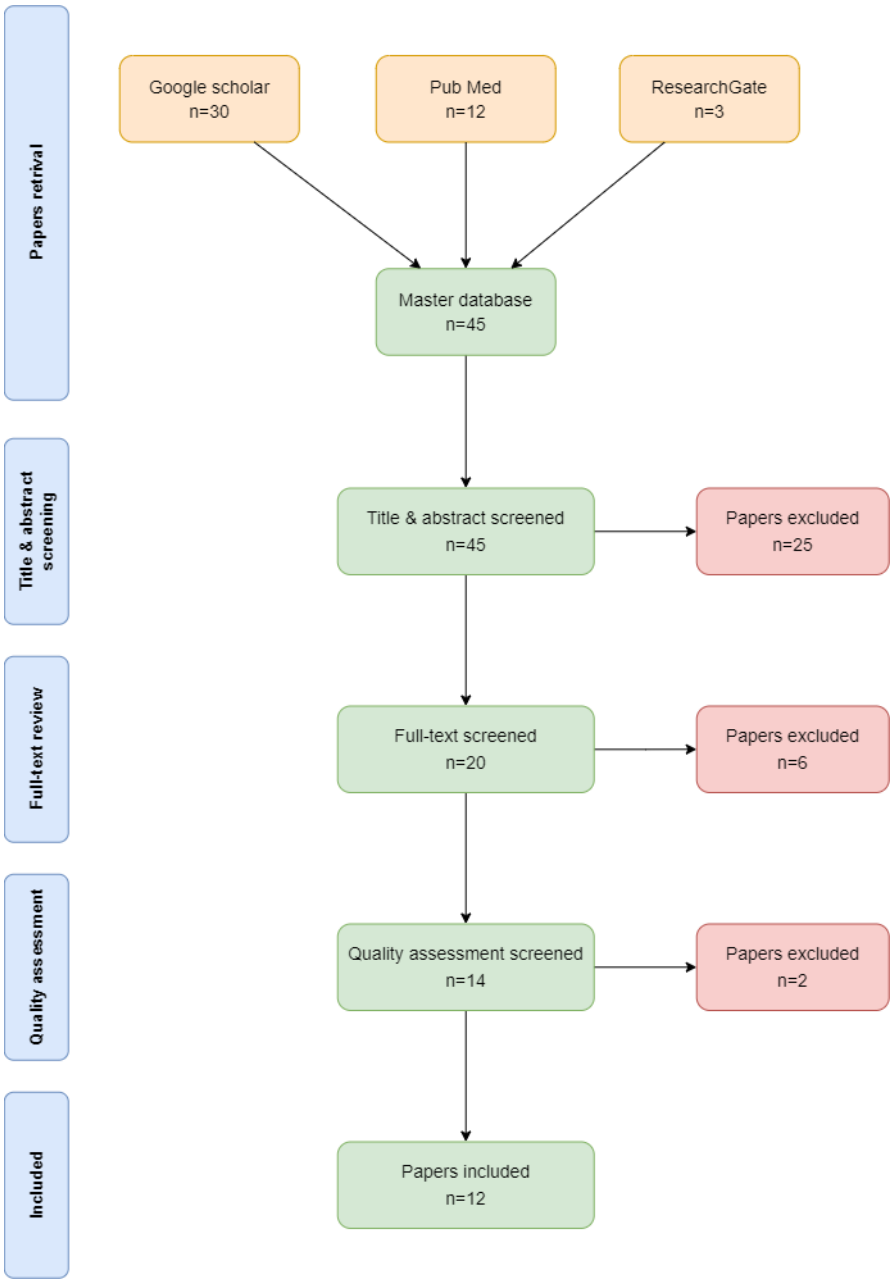


Figure 2.1: Selection process

At the initial stage, we tried to create a master database retrieving as much papers as possible. It was composed by 45 works and the provenience was shared as follows:

- 30 papers from Google scholar;
- 12 papers from Pub Med;
- 3 papers from ResearchGate.

After reviewing the titles and abstracts of 45 papers, 25 were excluded because they did not closely align with our research focus on informal caregivers having patients with dementia. The remaining 20 papers underwent a full-text screening, during which 6 more were excluded. Specifically, these papers either addressed unrelated medical conditions or generalized caregiving topics. The final 14 papers were then evaluated for quality, resulting in the exclusion of 2 more papers. These were dismissed due to unclear or poorly substantiated conclusions, as well as a lack of relevance to our study on informal caregivers of Alzheimer's patients. So the process included 12 papers in the final review which are listed in Table 2.1.

Paper names
Vivian Tran Louise I. R. Castillo and Thomas Hadjistavropoulos. Are mobile apps meeting the needs of caregivers of people living with dementia? An evaluation of existing apps for caregivers. Vol. 28. 2024. [6]
Marina Sala-González et al. Mobile Apps for Helping Informal Caregivers: A Systematic Review. Vol. 18. 2021. [10]
T. Halbach et al. Mobile application for supporting dementia relatives: A case study. Vol. 256. 2018. [1]
Milena Guessi, Amika Shah, and Emily Seto. Smartphone applications for informal caregivers of chronically ill patients: a scoping review. Vol. 5. 2022. [11]
B.H. Davis et al. Developing a pilot e-mobile app for dementia caregiver support: Lessons learned. Vol. 18. 2014. [9]
Lori Wozney et al. Commercially Available Mobile Apps for Caregivers of People With Alzheimer Disease or Other Related Dementias: Systematic Search. Vol. 1. 2018. [24]
Ellen Leslie Brown et al. CareHeroes Web and Android™ Apps for Dementia Caregivers: A Feasibility Study. Vol. 9. 2016. [4]
YunHee Shin et al. Effects of App-Based Mobile Interventions for Dementia Family Caregivers: A Systematic Review and Meta-Analysis. Vol. 51. 2022. [17]
Ellen Brown et al. Smartphone-Based Health Technologies for Dementia Care: Opportunities, Challenges, and Current Practices. Vol. 38. 2017. [5]
Taylor A. James, Dara James, and Linda K. Larkey. Heart-focused breathing and perceptions of burden in Alzheimer's caregivers: An online randomized controlled pilot study. Vol. 42. 2021. [20]

Paper names
Angel H. Wang et al. Beyond instrumental support: Mobile application use by family caregivers of persons living with dementia. Vol. 21. 2022. [22]
N.C. Tan et al. Patient-caregiver twinned mobile phone application to promote medication adherence. Vol. 33. 2024. [19]

Table 2.1: Selected papers

2.3 Selected Papers

After carefully selecting relevant papers, we can discuss the current state of the art regarding caregivers, with a particular emphasis on those caring for patients with Alzheimer's or other forms of dementia. First, we review what researchers have accomplished over the years. Then, we examine in detail the improvements made in recent years and finally we analyze studies that aim to understand the needs of caregivers.

The challenges faced by caregiver were summarized by Brown et al. [5]. The authors jumped into the critical role of smartphone technology in supporting caregivers of individuals with Alzheimer's disease and they highlighted the significant challenges faced by caregivers including stress, burden and lack of support. Given the increasing prevalence of Alzheimer's disease and the potential of smartphones to deliver healthcare, the authors emphasize the need for innovative solutions. As shown in Table 2.2, a complete review of existing smartphone apps designed for Alzheimer's disease caregivers reveals that the majority of these applications offer limited functionalities, mainly focusing on providing general information about Alzheimer's disease and caregiving.

The study also points out the inconsistency in the readability of app content, underlining the need for apps designed to accommodate individuals with varying literacy levels. In fact, if Alzeheimer's disease caregivers do not express enthusiasm about the use of smartphone apps, both doctors and caregivers may be reluctant to use apps as a component of care management. To address the complex needs of Alzheimer's disease caregivers, the authors propose the development of multifunctional apps that are user-friendly and capable of facilitating communication between caregivers and patients. The work is considered as a good starting point, but there is a lack about the name of the applications taken into consideration, so more works were considered also to understand what was the progression from the date of the publication of that paper to last days.

Marina Sala-González et al. [10] conducted a review of mobile applications designed to support informal caregivers. Their work offers a valuable overview of the existing landscape of such apps. While the authors presented a wide number of studies involving different type of patients, the subset focusing on Alzheimer's disease caregivers was closely examined, such as CareHeroes [4], Story-Call [9] and the work of Halbach et al. [1]. First consideration is that they all have 2 primary limitations:

Features	n (%)
Caregiver tips	7 (53.8)
Social networking capacity	6 (46.1)
Track appointments	6 (46.1)
Medication management	6 (46.1)
Alert or reminder capacity	5 (38.4)
Collection of care recipient clinical information	4 (30.7)
Links for community services	3 (23)
Feedback from health care professionals	2 (15.3)
Monitoring device	2 (15.3)

Table 2.2: Brown et al., features of caregivers app (n = 13) [5]

- **Short or middle usage periods:** this constraint can hinder the assessment of the app's long-term efficacy and impact on caregivers' well-being. In order to fully understand the benefits and challenges associated with these tools, it is essential to conduct studies that track their use over extended periods;
- **Small sample sizes:** this factor can compromise the statistical analysis of the research, making it difficult to achieve definitive conclusions about the apps' effectiveness.

Then, a key point to underline is that some of the applications consider also professional caregivers. It could be something very helpful since they are very experienced people on the sector of caregiving, but, on the other side, it could statistically influence the results since the act of caring is part of their professional life and so it could be subjected to some biased considerations.

By looking at the most recent literature works about the topic, Milena Guessi et al. [11] observed that smartphones is a promising method to support informal caregivers of chronically ill individuals. Through their search, summarized in Table 2.3, they analyzed the availability of native apps targeting caregivers in application stores and they noticed that academic literature on applications' design and effectiveness evaluation is limited [24]. In fact, most of the papers are centered on the development, usability or feasibility studies and only a little part of them are purely on effectiveness of what has been developed.

Application name	Android	iOS	Study type
CareHeroes	Yes	No	Feasibility
CAST	Yes	No	Feasibility
Cubes	Yes	Yes	Usability
Dea	Yes	No	Usability
Dementia Support for Carers	Yes	Yes	Development Protocol
FamTechCare	No	Yes	Effectiveness Feasibility Cost-effectiveness
MIT	No	Yes	Feasibility
Memory Board	Yes	No	Usability
Inlife	Yes	Yes	Protocol Development
mYouTime	Yes	Yes	Usability
PsyMate	Yes	Yes	Effectiveness
Story-call	Yes	No	Development

Table 2.3: Milena Guessi et al. partial table [11]

The problem that arises is that developing native apps are highly effective for leveraging particular sensors, but, on the other side, they may restrict the number of individuals who can test and evaluate them.

Another conclusion of the authors is that previous literature reviews have mainly focused on single chronic conditions or general-purpose apps. Clearly, general-purpose apps should be refined to have more personalization, but, on the other side, it is important to underline how different pathologies can have unique needs since the affected human area is different. For this reason, a specific category of caregivers can have different needs in terms of particular functionalities and support features. For example, caregivers for Alzheimer's patients may need tools for memory and behavior management, while those caring for individuals with chronic obstructive pulmonary disease may require apps for monitoring respiratory health and managing medications. Therefore, it is crucial to design apps that can be tailored to the specific requirements of various chronic conditions and the unique challenges faced by their caregivers. This can include personalized content, adaptable user interfaces and resources about specific conditions, which would make the apps more effective and user-friendly for different caregiving scenarios. In addition, there could be an high percentage of older people

that are not familiar with the use of technology [17] [20]. Consequently, app-based mobile interventions may represent a novel object of burden for caregivers and for this reason they have to be intuitive.

Other works were more focused on the personal domain of the user. For example, Wang et al. [22] focused their attention on apps adaptation to individual needs and minimization of the impact on the person and family. In particular, caregivers used apps in order to communicate both their specific needs and the ones of their loved ones. Apps were adapted for different purposes, including health monitoring, communication and scheduling. This personalization helped to make caregiving tasks more efficient and effective. In addition, they tried to minimize the impact of dementia on both patients and their families by facilitating reminders for medication and appointments, reducing the cognitive load on caregivers and helping them to maintain a routine and a structure in their daily life.

The work made by Castillo et al. [6] strenghtened all the previous considerations. In fact, the authors created an accurate analysis on the main features for the existing applications related to the Alzheimer's caregiving and talked to final users in order to better understand which are the features that they consider more important. In particular, they used 2 different applications and compared the results [28] [55]. The features found in the application are more or less the same of the Brown et al. study [5] and reported in Table 2.2, but caregivers contributed significantly to give a direction to the research. This is fundamental and it contributed in a significant way in understanding which is really important for the activity of caregiving. For example, adding the specific context in which the caregiver is living could help to optimize the application's customization and providing resources tailored to the unique needs of different subgroups of caregivers. For instance, caregivers dealing with aggressive patients need access to specific strategies in order to have control over them in an efficient way.

Participants highlighted both positive and negative aspects of the systems. On the positive side, users expressed high levels of satisfaction with the type of information provided and the visual appeal of the system, underlining the important role of the presentation layer in user experience. This feedback confirms the significant impact of an engaging and well-organized interface on user satisfaction. On the other side, the lack of customization options emerged as a critical limitation. This inadequacy may constrain the system's ability to meet different caregiver needs and potentially diminish its overall value. In addition, the navigation system has been another important point, with many users reporting that it was counterintuitive and challenging to use. This suggests that improvements in navigational design are necessary to enhance user accessibility and ease of use. It was also noticed that a lot of people asked for the coordination care management in order to share the overall stress caused by the caregiving activity. So, while the presentation layer is recognized for its effectiveness, addressing the issues related to customization and navigation could significantly improve the system's overall utility and user experience.

Generally speaking the previous findings highlighted that the value of apps goes beyond their basic functionality and their ability to help with care provision since they are also able to promote richer interpersonal connections, enhancing personhood and sustaining family routines.

Finally, even if it does not refer to works about Alzheimer, Tan et al. [19] explored the use of NFC technology in a healthcare environment. This study focused on evaluating the feasibility, acceptability, utility and impact on glycaemic control of the MediEasy app for patients with type-2 diabetes mellitus and their caregivers. In

Chapter 3, we see more in detail how this work inspired the creation of a feature, but we also discuss the reasons why this was ultimately not implemented.

2.4 Applications in Apple and Google Play stores

In order to conduct a complete analysis of applications relevant to our context, we have expanded our search to include both the Apple App Store and Google Play Store. The primary goal of this search was to identify a different set of applications, evaluate their features and obtain users' feedback to include the most relevant implementations. By analyzing these feedback, we aimed to gain insights about the most effective and innovative features and take inspiration for the definition of the product (as it will see in Chapter 3).

It is important to underline that also other applications with different targets were considered. The goal was to discuss with experts to determine whether these features, despite being designed for different purposes, could potentially be useful in our use case.

The results of our search are summarized in Table 2.4. The table contains the availability of each application on Android and iOS platforms, the number of reviews each application has received and their average ratings.

Application name	Android	iOS	Reviews	Average evaluation
AlayaCare	Yes	No	39	2.7/5.0
Evercare	Yes	Yes	-	-
Ianacare	Yes	Yes	-	-
Homecare	Yes	Yes	-	-
Homage	Yes	Yes	-	-
CareApp	Yes	No	-	-
Hibi	Yes	Yes	-	-
AxisCare	Yes	No	-	-
BlueSky Mobile	Yes	Yes	1	5.0/5.0
Caregiver Mobile	No	Yes	-	-
CareLinx: In-Home Care	Yes	Yes	19	4.3/5.0
CareConnect	Yes	No	15	4.8/5.0

Table 2.4: Applications on the App and Google Play stores

The data presented in Table 2.4 reveal several key points about the applications taken into consideration. In fact, the majority of applications are available on both Android and iOS platforms, indicating a broad market reach. However, there is a significant lack in the availability of user feedback. Actually, only a subset of applications have

users' reviews and the quantity of reviews available is insufficient to comprehensively assess the strengths and weaknesses of these applications. In the absence of detailed users' feedback, our analysis focused on the descriptions and screenshots provided by the applications on their respective stores. We carefully examined these elements to extract and identify the most promising features and functionalities highlighted by the developers. We explore some of the applications and provide a brief description of their functionalities.

AlayaCare

AlayaCare app makes it easy for caregivers to access and track patient health and fitness. Real-time scheduling, time tracking, charting and clinical documentation are available, so the caregivers can deliver higher-quality health care service. Figure 2.2 shows some screenshots of the application.

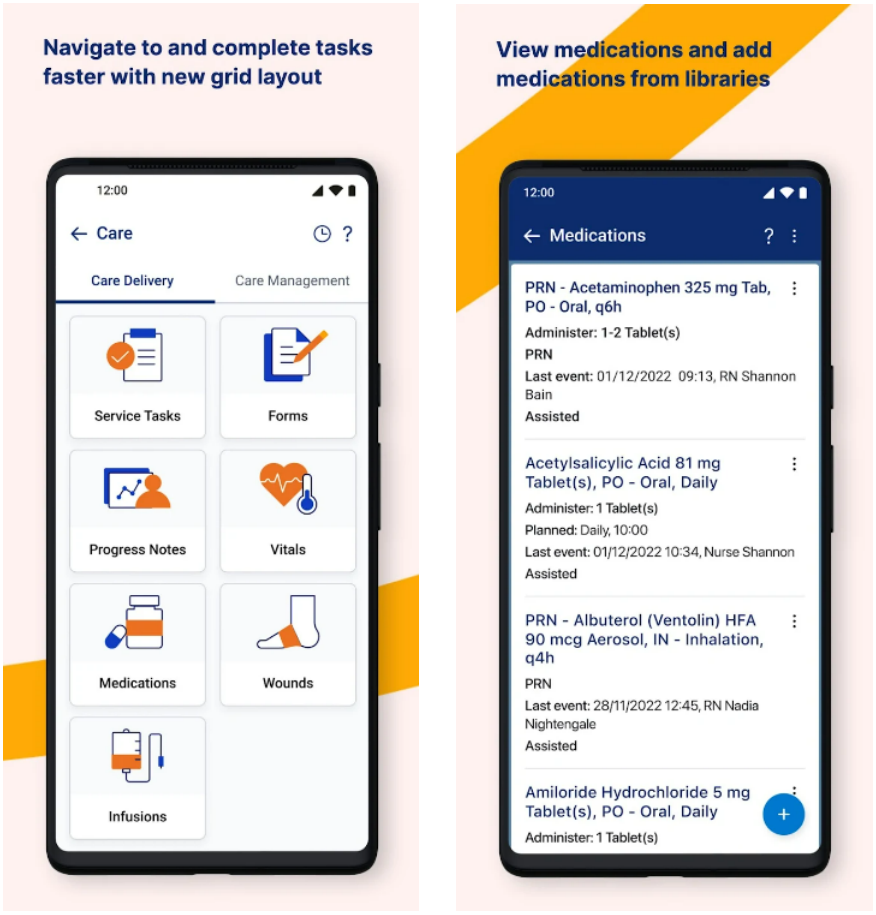


Figure 2.2: AlayaCare screenshots [30] [31]

The app is available in English, French and Spanish and provides these functionalities:

- real-time access to schedules, route details, billing, safety, time tracking, patient

data and form reporting;

- update and track patient visit including progress notes, medications and tasks;
- track data accurately with GPS-based clock in/out and location-based electronic visit verification;
- see updated schedules in real time, then stay on track with mobile alerts and reminders;
- keep data secure with compliance with privacy legislation.

A lot of reviews underline that there are a lot of bugs and many problems regarding the navigation and user experience, degrading the user satisfaction for the application.

Ianacare

Ianacare (i.e., I Am Not Alone Care) is an integrated platform for family caregivers that organizes and moves all the layers of support. It helps to coordinate activities with friends and family, utilize employer benefits, discover local resources and get personalized guidance. The mission of the application is to encourage, empower and equip family caregivers with the tools and communities, so no caregiver does it alone. Figure 2.3 shows some screenshots of the application.

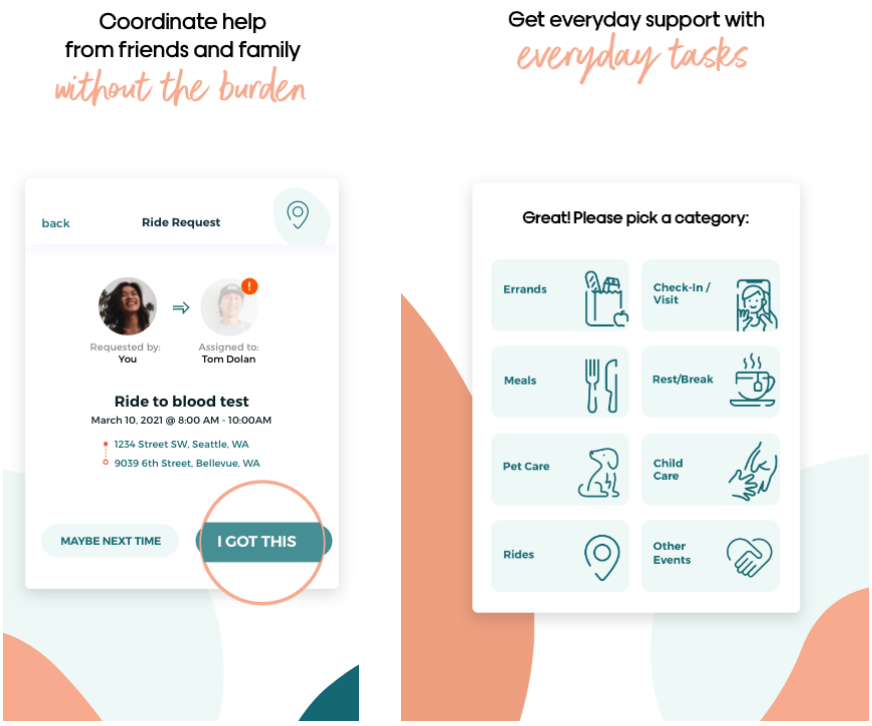


Figure 2.3: IanaCare screenshots [74] [75]

It provides these key features:

- receiving practical help, sharing care requests with the team to get practical support with meals, check-ins, rides, etc;
- easy team creation inviting friends, family, neighbors, co-workers, community members, professional caretakers and anyone else who wants to help;
- caregivers can post in a private ianacare feed and everyone on the team can share news, offer support and get updates on the care of caregivers' loved one, keeping everyone up-to-date;
- organizing with a team calendar, so that every task that is requested shows up on caregiver's team calendar. In this way, he or she can stay organized and know exactly when people plan to help and where the caregiver need additional support;
- possibility to control requests, notifications and updates and how the caregiver get them like email, SMS, push notifications.

Hibi

Hibi is designed by families, for families and empowers families managing, coordinating and navigating their child's health journey. Made in partnership with child health and care experts, it is the only app a parent need to track, record and navigate a child's care. Users use Hibi to manage their loved ones' care, supporting people with: neurodiversity, Down's syndrome, developmental delays and disabilities, genetic disorders, rare diseases, diabetes, epilepsy, mental health conditions and more. Figure 2.4 shows some screenshots of the application.

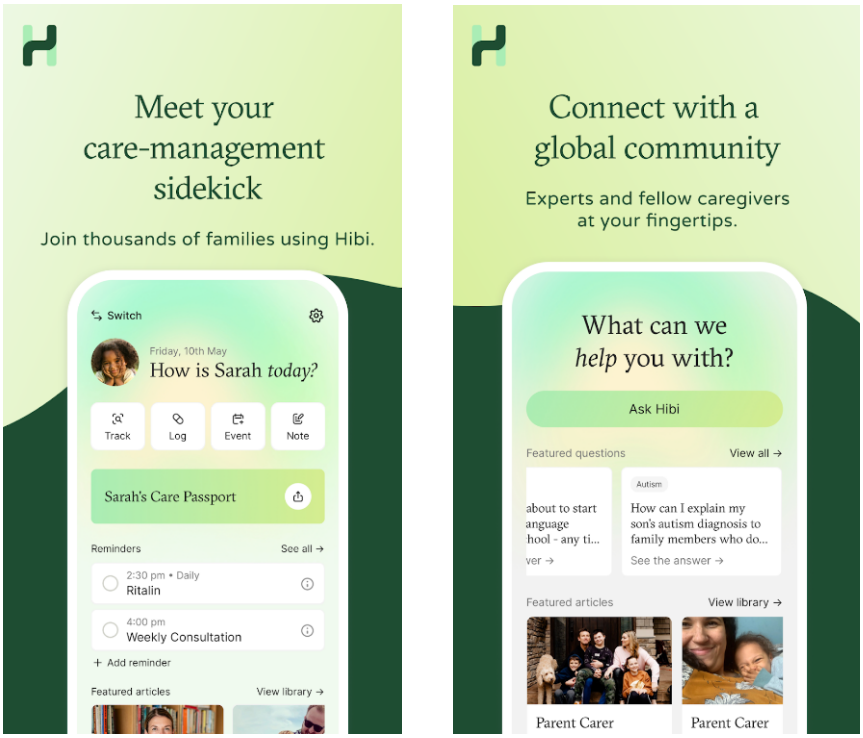


Figure 2.4: Hibi screenshots [72] [73]

It provides the following services:

- log physical symptoms, behavioural moods and get personalizations for the patient by adding the own custom symptom trackers;
- set up a medication plan, get useful reminders; to help the caregiver in remembering events and keeping track of medicines and how they make them feel;
- store patient health history and view it easily in one place. Record diagnoses, past injuries, allergies or intolerances and upload or scan important health care documents;
- allow family members and health professionals to see child's health symptoms, appointments, medication intake and behaviours by inviting family and health professionals to the app;
- access personalised content and information tailored for the caregiver and his family.

Even if the application is thought for children's care, its features could be decline to the Alzheimer's patients.

Caregiver Mobile

The Caregiver mobile app works in tandem with Belle X mPERS devices to offer increased peace of mind to subscribers and their loved ones. In this way, caregivers can have a major control over the patient. Figure 2.5 shows some screenshots of the application.

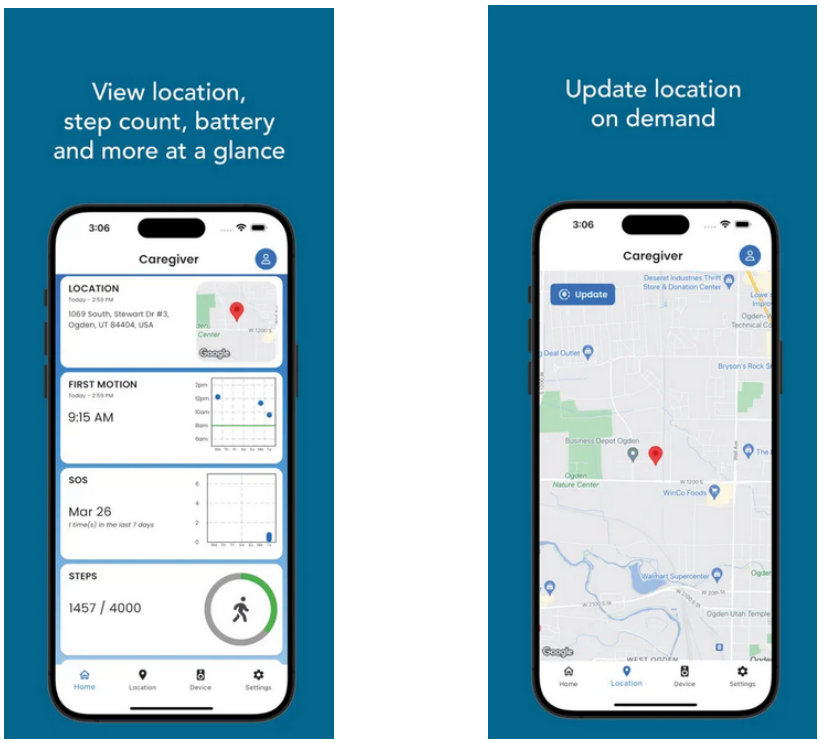


Figure 2.5: Caregiver Mobile screenshots [48]

By using the app, caregivers can:

- locate the device on demand;
- ring the device to help find misplaced units;
- receive app notifications for button presses, falls, low battery, power off, and first motion of the day;
- view button press history;
- check device battery level;
- track steps and set a daily step goal.

One of the main problems of this application is the lack of real useful functionalities, but, on the other side, it could be interesting for tracking the position of the patients.

This qualitative assessment allowed us to identify common themes that could potentially influence user experience and satisfaction. Our analysis aimed to correlate these identified features with findings from the scientific literature on similar applications. Our goal was to bridge the gap between practical observations and established research, ensuring that our conclusions are well-grounded in both real-world user experiences and academic studies. This approach enforces our understanding of key factors that drive user satisfaction.

2.5 Lessons learned

Research indicates that eHealth interventions can positively impact caregivers' self-efficacy, competence and depression. By looking at both analyzed applications and papers, these interventions include online support groups, educational resources and telehealth consultations that provide caregivers with the necessary knowledge and emotional support to manage effectively their caregiving responsibilities. Despite these promising aspects, robust data on the overall effectiveness of these online tools remains limited. There is a significant need for further research to comprehensively determine their benefits in caregivers' daily lives. Studies should explore how these interventions can be optimized to provide the greatest support, including examining different formats, frequencies and types of content that are most beneficial [7] [13]. Early detection of mild cognitive impairment symptoms and the provision of supportive tools are crucial for delaying its progression to various forms of dementia.

Chapter 3

Product definition

In this chapter, we provide a comprehensive description of the *NetCare* application's functionalities. These ones have been carefully defined based on the extensive work carried out in the previous chapter, where we undertook a in-depth analysis to identify and choose the key features that would be most beneficial for our specific case of study. This ensured that the proposed functionalities were in sync with the overall goals of our study. Additionally, we engaged in thorough discussions with experts in the field to validate our proposals from multiple perspectives. These consultations were important in terms of refinement as they provided valuable insights that helped to confirm the practicality and relevance of the identified features.

Moreover, we carefully describe each functionality, explaining how it works and its potential impact on the overall effectiveness of the *NetCare* application. Through this detailed exploration, we aim to provide a clear understanding of the application's features and how they are expected to contribute in order to achieve the desired outcomes in our specific context.

Finally, we provide an explanation of several features that were initially considered, but they were ultimately excluded from the final design. We briefly describe them and explain the reasoning behind their exclusion.

The following features are presented:

1. Authentication;
2. Coordination;
3. Notes;
4. Caregiver Tips;
5. Patient Tips;
6. Profile.

It is important to underline that the entities included are the caregivers and patients.

3.1 Authentication

Authentication is a basic feature which ensures only authorized users have access to certain functionalities or data. The authentication process typically involves two main components, so called login and signup. This section explains how these components work within the application.

Login

The login allows users who have already registered in the application to access to their accounts. The process involves the following steps:

1. **User input:** users are prompted to enter their credentials *email address* and *password*;
2. **Verification:** application checks the entered credentials in the database;
3. **Access granted or denied:**
   - if the credentials are correct, users are granted access to their account and they are redirected to the application's home page;
   - if the credentials are incorrect, users are notified and they are prompted to try again.

For simplicity, a password recovery option has not been implemented at this stage, but the final product should include this feature.

Signup

The signup feature allows new users to create an account within the application. This process includes the following steps:

1. **User registration:** users are asked to provide necessary information in order to register, which are:
   - name;
   - surname;
   - email;
   - password;
   - whether is caregiver or not;
   - One-Time-Password code.

   In particular, the password has to be retyped in order to create a double check. The *One-Time-Password* is a 6-digits code which is given by the administrators of the applications, so they can be sure that only authorized users can subscribe to this application. This was inserted only to facilitate the testing and it should be removed in case of publication in the app stores. It is important also to underline that being a caregiver or not can limit the available functionalities for the account;

2. **Data Validation:**
    - the application checks if the *email address* is already in use. If so, user are prompted to choose another one;
    - the application checks if the *One-Time-Password* code is valid. If it is not, users are prompted to rewrite the correct one;
    - the password is validated to ensure it meets the required security standards and double checked.
3. **Account Creation:** if the provided data are valid, the application creates a new user account and stores the credentials in the database;
4. **Login Access:** once the account is successfully created, users automatically log in with their credentials.

3.2 Coordination

The *Coordination* feature is one of the most important of the entire application. It is one of the most requested features as shown in Chapter 2.3. First we describe the feature from the caregiver's point of view and then we highlight the variants for the patient.

The idea is based on 3 main concepts which are the following:

- **Room:** it is a group which should ideally include one patient, the caregiver who created the room and all other caregivers;
- **Event:** it can be assigned only to caregivers and includes a description and some optional media; in particular, this can be self-assigned by the creator or it remain as a pending event;
- **Activity:** it is assigned exclusively to patients and it includes a description and media; in addition, this can be marked as executed by the patient and caregivers are notified about it.

In order to better understand the difference between an event and an activity, we make an example. For instance, taking the patient to the doctor could be considered an event since the patient is passively involved in this occurrence and he or she does not have to inform the caregivers about anything. In fact, it is just something between caregivers. Let's consider taking a pill instead. This could be considered as an activity since the patient is actively involved in this occurrence and he or she has to inform the caregivers about the completion of the activity.

More deeply for what concerns the *room*, we have that:

- all the user can be in one and only one *room*;
- only caregivers can create a *room*;
- there can be one and only one host in a *room*, who can have also additional functionalities.

If the caregivers are not actually in a *room*, they can create a new *room* or they are put in a Waiting *room* staying up for any invitation from some other *room*'s host. The user, who is already in a *room*, is able to see the following:

1. **Family:** who is part of the *room*;
2. **Calendar:** the shared calendar between the components of the group;
3. **Pending events:** the pending events which are not assigned to any user yet.

Clearly, each user can exit the *room* and the host can also cancel it with effects also on the participants.

3.2.1 Family

In this part of the application, cards are displayed representing the individuals who have currently entered the *room*. Each card contains all the information the users provided during the authentication process. Additionally, the *room*'s host has the option to expand the participant list by inviting new users. This can be done by adding their email addresses, allowing them to join the *room* and participating in the ongoing activities.

3.2.2 Calendar

The app includes a calendar which shows a list of events scheduled for the selected day. Users have the flexibility to create new events or activities straight within the calendar. They can also modify existing entries, making it easy to update details such as dates, times or assigned user. The cards created for a specific event highlight the starting and ending date, starting and ending time and the user for which the event is assigned. This allows to better balance the work among caregivers. Additionally, users can expand individual events to access more detailed information, including descriptions and media. During the creation, the user can decide to create a repetitive events or activities. This feature ensures that all relevant details are readily available, helping users to manage their schedules more effectively and stay organized during the day.

3.2.3 Pending events

This final part of the application can be considered as a shortcut for the caregivers to get the pending events, i.e., the ones which are not assigned to any caregiver. This improves the event management for caregivers, enabling them to easily identify and take charge of tasks that require immediate attention. By providing a centralized location for unassigned events, the application ensures that nothing is neglected and caregivers can efficiently manage their responsibilities. This ultimately leads to better coordination and more effective care as already explained in Section 3.2.2.

3.2.4 Variants for patients

In this section we discuss about the variants for the patients. In fact, they cannot have the same rights like a caregiver. For this reason, there are some differences between what a caregiver and patient can do. Some of them are already mentioned previously:

- a patient cannot create a *room*;
- a patient is not allowed to add users in the *room*.

Additionally, *"Pending Events"* are replaced with *"Activities"*. This change is useful for patients to notify caregivers when an activity was completed. When a patient finishes an activity, they can easily send a notification to all caregivers within the *room*, informing them about the completion. This feature not only keeps caregivers updated in real-time, but also promotes better communication and coordination and it could enhance caregiver's well-being.

3.2.5 Coordination use cases

In this section, we present two main examples which allow us to understand how the coordination system works.

Let Bob, Alice and Charlie be three caregivers and let Dave be an Alzheimer patient. All the caregivers have their own life with family's commitments, but they have to assist to Dave because he is not able to remember things because of the disease.

Dave has a lot of activities to remember like taking pills and drugs and going to the supermarket near home. On the other side, the three caregivers have to manage all the commitments related to Dave like taking him to the doctors or other places since he does not have the driving license because he did not pass the health test.

In the following section, we explain an example of events management and activities completion.

Events management

During this week, Dave has to be taken to the doctor and to the gym once a week at the same day, so Bob creates the events associate to these commitments. By looking at his personal appointments, he is able to take Dave to the doctor, so he creates and assigns the activity to himself.

On the other side, Bob sees that he cannot take Dave to the gym because he has to take her daughter to tennis lessons. For this reason, he decides to create a recurrent pending event for the next 2 weeks in order to let any caregiver accept the commitment.

Alice, thanks to a daily notification, opens the application and checks for pending events to do. She sees that Dave needs to be taken to the gym and she decide to accept the commitment for this week.

Then also Charlie opens the application thanks to the daily notification. By opening the calendar, she sees that Bob and Alice have already taken some activities in charge, so she decides to see whether there are any activities or not the next week. Since Dave needs to be taken to the gym once a week, she decides to accept the event in order to divide the events equally with Alice and Bob.

Figure 3.1 shows the flowchart for events management.

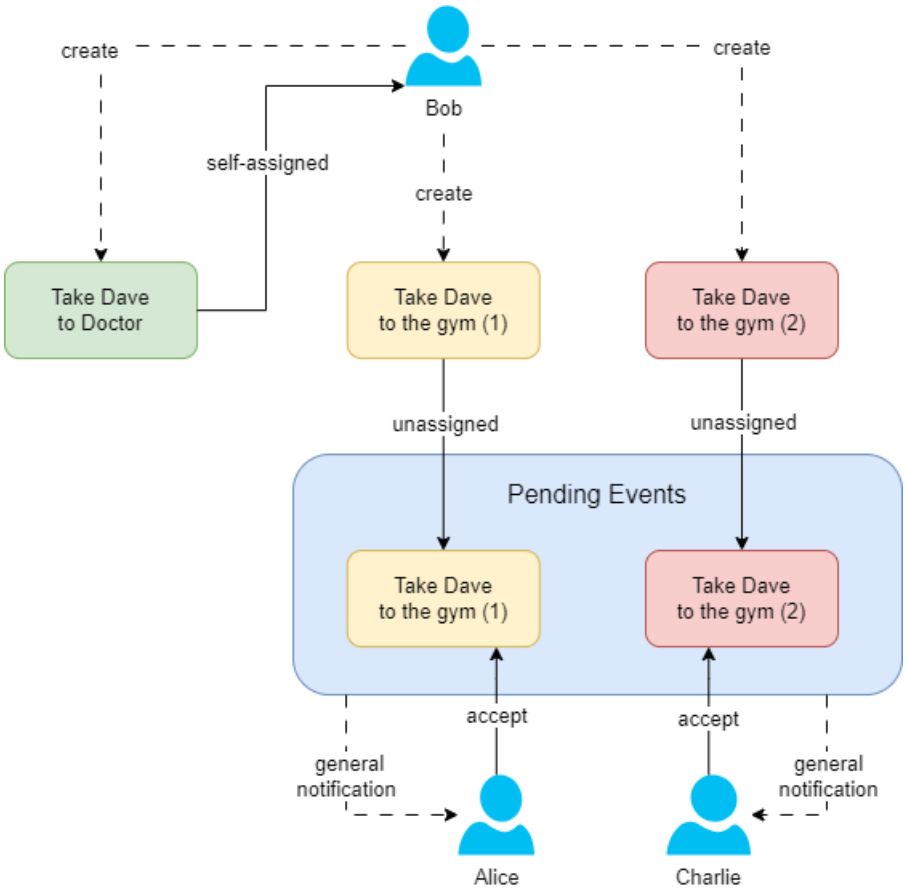


Figure 3.1: Events management flowchart

Activities completion

During this week, Dave has to take some drugs each day for a treatment. He should take them in the periods suggested by the doctor, so Alice creates three daily recurrent activities assigned to Dave for this week.

It is evening and Dave always has dinner at 6.30 PM. Before having it, he should take the drug, but he forgets about it. Bob opens the application after finishing work and he spots that he has not received any notification about Dave's drug, so he decides to call him to remember about it.

The next day, Dave remembers correctly and he takes the drug. After this he marks the activity completed, so all the three caregivers are warned about the completion of the activity.

Finally Dave takes the drug a little bit late once, but he completed the treatment without any problem.

Figure 3.2 shows the flowchart for activities completion.

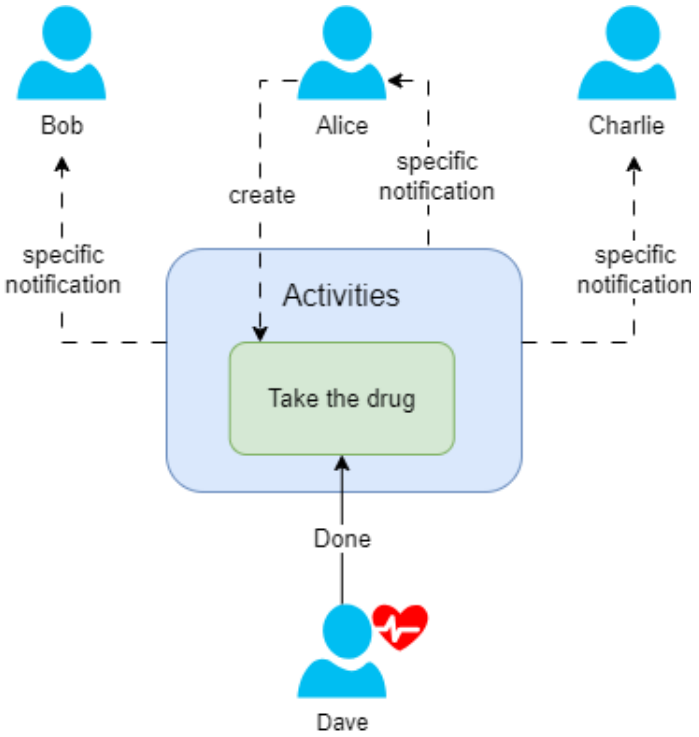


Figure 3.2: Activities completion flowchart

It is important to remember that both events and activities can be edited and deleted, making the coordination feature more flexible and completed.

3.3 Notes

This features allows the users to create their personal notes. In particular, each note has:

- **Hashtag:** it is used to identify the category of the note. For simplicity, each note can have one and only one hashtag;
- **Title:** it represent the concept express by the note;
- **Colour:** it can be changed thanks to a colour picker and helps to better classify the notes;
- **Content:** it contains what the user want to write. It can contain also photos in the middle of the text;
- **Last Time Modified:** it is the last time the note has been modified. Clearly, an interaction, which does not modify the note, does not modify either this value.

As for the calendar events, users can create new notes to save important information. For instance, if users are doing a consulence with a doctor or they have seen an interesting gym offer, they can save information or photos as a memo and maybe they can use it later in a discussion with other caregivers.

They can also modify existing ones, making it easy to update details mentioned above and expand individual events to access more detailed information, including descriptions and media.

In addition, if users are in a *room*, they can share the selected note with a subset of other *room* members. Shared notes are in read-only mode, meaning they cannot be edited by others, but they can be further shared with additional users within the *room*. This functionality enhances collaboration while maintaining control over the content.

Since the list of the notes could be very long, in order to facilitate the retrieval of a note, search functionality is implemented and only notes with the correspondent hashtag are displayed.

3.4 Tips

The tips functionality provides two categories of tailored advice:

- Caregiver tips;
- Patient care tips.

To ensure personalized guidance, users can apply specific filters for a quick and efficient search of relevant tips. For example, in the caregiver tips, the user can select the duration of the tip (e.g., 10 min, 15 min, etc.) and the category (e.g., meditation, excercise, etc.); in the patient care tips, the user can select the phase of the patient (e.g. phase 1, phase 2, etc.) and the category (e.g., anxiety, nutrition, etc.). Each tip is presented as a card that can be expanded to reveal the full content, allowing users to easily access and read detailed advice. The idea is trying to help the caregiver to create a relaxed environment and manage the patient's behaviour in case of difficulty.

In order to stimulate the usage of this section a caregiver tip and a patient care tip are displayed in the home screen. Since the application's primary goal is to alleviate stress and support caregivers, tips are mainly adressed to them.

3.5 Discarded features

The initial proposal contained other features, but they were discarded. In particular, we summarize them as follows:

1. **NFC notification:** very often patients need to perform specific physical actions, such as taking medication or retrieving a pill, so it is important to ensure these tasks are completed accurately and in time. To facilitate this, NFC technology can be implemented as a monitoring and notification system. For example, similar to the approach detailed by Tan et al. [19], NFC tags can be strategically placed on objects or within the patient's environment. When the patient interacts with the tagged item, such as by scanning the NFC tag with a mobile device or a wearable, a notification can be automatically generated to confirm the action has been completed. This method not only enhances the responsibility of the patient, but also alleviates the burden on caregivers by automating the monitoring process. In Figure 3.3, it is represented in a detailed way how Tan et al. did the work, which could be declined in a similar way to the Alzheimer's patients and their caregivers;

2. **Health documentation updates:** this feature would involve creating a centralized, digital bucket where all clinical documents related to the patient are stored and regularly updated. In fact, maintaining accurate and up-to-date health records is fundamental for providing high-quality patient care. The bucket would serve as a dynamic repository for patient information, including medical history, treatment plans, medication records, test results and any other relevant health data;

3. **GPS localization:** it would involve the tracking of the patients in order to allow the caregivers to help them in case they are lost without realizing it. For Alzheimer's patients, it is often recommended to establish and maintain a consistent daily routine, which might include activities like taking a regular walk or going to the supermarket at the same time each day [33]. A well-established routine helps to reduce anxiety and confusion in both caregivers and patients, providing them with a sense of stability. However, there may be instances where the patient unintentionally deviates from this routine, which can lead them disorientation. In such cases, it would be highly beneficial if caregivers could be alerted when the patients step outside of their usual routine. For example, if a patient unexpectedly leaves the house at an unusual time or takes an unfamiliar direction, the tracking system could notify the caregiver immediately. This early warning would allow the caregiver to intervene rapidly, ensuring the patient's safety and preventing potential harm.

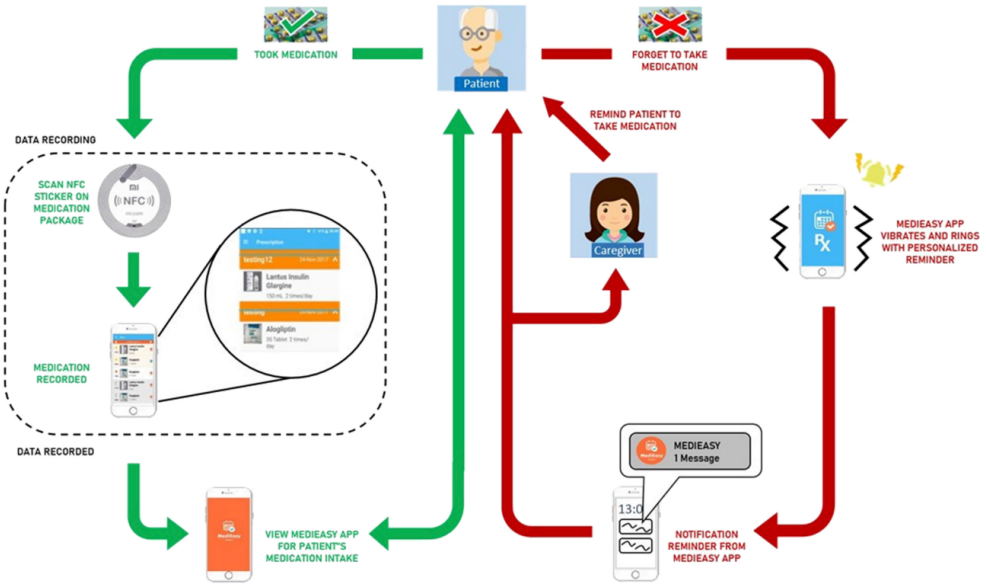


Figure 3.3: NFC implementation by Tan et al. [19]

After extensive consultation with experts in the field, we initially recognized the potential value and relevance of the features for our case study. However, after further consideration, we decided not to implement these features due to concerns related to the European Union General Data Protection Regulation (GDPR) [27].

The GDPR imposes stringent requirements on the processing and protection of personal data, in particular for sensitive health-related information. The implementa-

tion of NFC technology for monitoring patient actions and maintaining a centralized health documentation system could potentially expose sensitive data to risks that may violate GDPR regulations. For instance, the NFC-based system, while effective in tracking patient activities, could lead to the unauthorized collection or sharing of personal data. Similarly, a centralized digital repository for health documentation could present vulnerabilities in data security, increasing the risk of unauthorized access or data breaches. In addition, the use of GPS localization for tracking patients presents further challenges. GPS tracking involves continuous monitoring of a patient's location, which could lead to the unintended or unauthorized sharing of their movements and daily routines. Such data, if compromised, could be misused, posing significant privacy risks to the involved individuals. Given the complexity and sensitivity of these issues, which include not only data protection, but also the need for compliance with other regulatory commissions, we opted to take a more cautious approach.

As a result, we have decided to simplify the notification system as outlined in Section 3.2. So that, we have abandoned the health documentation update feature to avoid potential risks associated with handling sensitive health data. By simplifying these aspects of our system, we aimed to balance the need for technological innovation with the importance of data privacy and protection. However, since the features were considered valid, they are left as future works for improving the application, as we will see in Chapter 6.

Chapter 4

Technologies

In this chapter, we focus the attention on the technologies used to develop the project. We describe them and motivate their usage.

4.1 Android Studio

Android Studio [35] is the official Integrated Development Environment (IDE) for Android application development, created by Google. It is built on JetBrains' IntelliJ IDEA software and provides a robust platform for developers to build, test and debug Android applications efficiently. Figure 4.1 shows its official logo.

Figure 4.1: Android Studio logo

Android Studio integrates all the necessary tools for Android development in one place. This includes a code editor, a built-in emulator and a suite of tools for performance profiling and debugging. Some of the key Android Studio features are summarized as follows:

- **Support for multiple languages and frameworks:** used mainly for Android development, Android Studio also supports Flutter and other frameworks through plugins. This makes it a versatile tool for developers working on multi-platform projects;
- **Gradle build system:** Android Studio uses Gradle for its build system, offering flexibility and customization for building, testing and deploying applications. This system can manage dependencies, define build variants and perform automated tests;

- **Version control integration:** the IDE supports integration with version control systems like Git, allowing developers to manage their code versions and track changes;
- **Extensive documentation and community support:** Android Studio is supported by good documentation and a large community. Resources include official guides, tutorials and forums where developers can find solutions to common issues and share knowledge.

From a practical point of view, Android Studio was chosen because its emulator configuration was faster and simpler compared to other IDEs, such as Visual Studio Code. In addition, the resources used for Android Studio were less than Visual Studio Code in this PC configuration.

4.2 Dart

Dart [53] is an open-source, general-purpose programming language developed by Google, designed to build web, server, desktop and mobile applications. Figure 4.2 shows its official logo.

Figure 4.2: Dart logo

First introduced in 2011, Dart aims to provide developers with a language that is easy to learn and use, offering at the same time the performance needed for complex applications. It is particularly known for its use in the Flutter framework, which enables the development of cross-platform applications. Some of the key Dart features are summarized as follows:

- **Ahead-of-Time and Just-in-Time compilation:** Dart supports both Ahead-of-Time and Just-in-Time compilation. Ahead-of-Time compiles Dart code into optimized machine code before running the application, ensuring optimal, consistent production performance and efficient memory usage, ideal for mobile devices. Just-in-Time compiles code at runtime, allowing fast and flexible development [37];
- **Strongly typed:** Dart is a statically typed language, which means that type checking is performed during compile time. This helps catching errors early in the development process;
- **Asynchronous programming:** it includes robust support for asynchronous programming with features like async/await/yield, making it easier to handle operations that take time, such as database requests;

- **Rich Standard library:** it offers a complete standard library that includes core libraries, collection libraries and more, which helps developers to write more expressive and efficient code.

Another reason why Flutter was chosen is for a subjective preference to strongly typed programming languages and a strong knowledge of C++ which is very similar for what concerns syntax.

4.3 Flutter

Flutter [64] is an open-source UI software development toolkit created by Google. It is used for building natively compiled applications for mobile, web and desktop from a single codebase. Figure 4.3 shows its official logo.

Figure 4.3: Flutter logo

Released in May 2017, Flutter allows developers to create visually appealing and high-performance applications with the help of its rich set of pre-designed widgets. It is written in the Dart programming language, which is optimized for fast app execution and has a clean and easy-to-understand syntax, providing a robust framework. Some of the key Flutter features are summarized as follows:

- **Single codebase:** it enables developers to write code once and deploy it across multiple platforms, including iOS, Android, web and desktop, reducing development time and effort. This can be useful when fast design and a MVP (Minimum Viable Product) is needed;
- **Hot reload:** this feature allows developers to see the results of code changes in real-time without restarting the application, speeding up significantly the development process;
- **Rich widget library:** it offers a complete library of customizable widgets, making it easier to create complex UIs that are both attractive and functional;
- **Performance:** by compiling directly to native code, Flutter achieves high performance, ensuring smooth animations and a fluid user experience.

Flutter's popularity has been growing rapidly due to its efficiency and the strong community support around it. It is widely adopted by developers and companies to build applications with a consistent look and feel across different platforms. The choice was influenced also by the chacteristics of the programming language used.

4.4 Firebase

Firebase [56] is an app development platform created by Google that provides a variety of tools and services to help developers to build high-quality apps, letting their businesses growing up. Figure 4.4 shows its official logo.

Figure 4.4: Firebase logo

Originally launched as a backend-as-a-service in 2011, Firebase has evolved into a versatile platform that supports web, iOS, Android, Flutter and even Unity-based apps. Firebase is used across various industries for different purposes, such as building MVPs, managing app growth and enabling real-time communication. Its ability to provide scalable infrastructure, along with robust analytics and tools, makes it popular among startups and companies. It is particularly well-suited for projects where real-time data synchronization, serverless architecture and cross-platform support are crucial.

In this project, Firebase was used in order to speed up the development trying to achive as more features as possible. In fact, the development of a custom backend would have implied a higher time consumption and this saving was used to develop high-quality, fancy user interface. In addition, it is important to underline that Firebase backend can be replaced with any other custom backend thanks to the level of abstraction of the application (as it will discuss in Chapter 5).

4.4.1 Authentication

Firebase Authentication [57] is a backend service developed by Google to facilitate authentication in mobile and web applications. It offers the process of authenticating users, ensuring both security and ease of use for developers and end-users. Figure 4.5 shows its official logo.

Figure 4.5: Firebase Authentication logo

At its core, Firebase Authentication supports a variety of authentication methods including email and password, phone numbers and other identity providers such as Google, Facebook, X and GitHub. This flexibility allows developers to capture a wide audience with different login preferences.

The service is designed to be highly secure, implementing best practices for data protection and user privacy. It handles complex authentication tasks such as managing password resets, account linking and multi-factor authentication, basically removing responsibility on significant security concerns from developers. Moreover, Firebase Authentication integrates with other Firebase services, such as Firestore and Firebase Realtime Database, enabling developers to build robust, feature-rich applications quickly.

In order to simplify the development and the possibility to manage the access to the application, only email and password authentication method was implemented with a custom *One-Time-Password* (as it will discuss in Chapter 5).

4.4.2 Firestore

Firebase Firestore [59], part of Google's Firebase platform, is a cloud-based NoSQL database that provides a scalable and flexible solution for storing and syncing data in real-time. Figure 4.6 shows its official logo.

Figure 4.6: Firebase Firestore logo

It is designed to handle the complexities of modern app development by offering an ideal integration with other Firebase services, such as Firebase Authentication, as well as robust support for mobile and web applications. Firestore allows developers to store data in documents, which are organized into collections, facilitating a highly structured and dynamic data model.

One of the features of Firestore is its real-time data synchronization. Changes to data are reflected instantly across all clients connected to the database, ensuring that users always have access to the most up-to-date information. This is particularly important for applications that require live updates, such as collaborative tools and and real-time dashboards.

Then Firestore's query capabilities enables developers to retrieve data efficiently using queries. It supports indexing and compound queries, allowing for quick and responsive data retrieval even from large databases. Moreover, Firestore is built with a scalability-first approach with the idea of satisfying the needs of both small and large applications.

Before choosing the database, an accurate research to see the key difference between Real-time database [61] and Firestore database [59] was done. The reasons why

Firestore was chosen over Real-time database are the following:

- **Data model:** the document-based model allows for better data organization. The hierarchical structure of collections and documents makes it easier to make application's models and perform complex queries on the fields of a particular document;
- **Query capabilities:** it supports a more powerful and expressive querying system than the Realtime Database. It is possible to perform compound queries, use indexed queries to retrieve data quickly and filter data based on multiple fields. This makes Firestore much more efficient and capable when dealing with large databases and complex querying requirements. In fact, on Real-time database the queries are way more limited since there is not the document, but pure hierarchical structure;
- **Google Cloud integration:** it integrates ideally with other Google Cloud services and Firebase products. This allows for a cohesive development experience and enables to including features like Cloud Functions and Firebase Authentication.

For these reasons, Firestore appeared to be an option which was better suited to the structure of this project [62].

4.4.3 Storage

Firebase Storage [63] is a cloud-based storage solution offered by Google as part of its Firebase platform. Figure 4.7 shows its official logo.

Figure 4.7: Firebase Storage logo

It is designed to help developers securely store and serve multimedia content, such as images, videos, audio, and other types of files. By leveraging robust and scalable Google's infrastructure, Firebase Storage allows developers to handle large volumes of data, ensuring that the stored content is accessible with low latency and high reliability. This makes it particularly well-suited for applications where quick access to media files is very important, such as content sharing platforms or online learning environments. Additionally, Firebase Storage's automatic handling of resumable uploads, including retry mechanisms for unreliable network conditions, simplifies the process of uploading large files, ensuring that the operation can be completed even in challenging network environments.

Firebase Storage uses Google Cloud Storage under the hood, providing the same level of security, redundancy and scalability that other Google's services have. Developers

can easily upload and download files directly from the client or through the intuitive Firebase SDK.

4.4.4 Messaging

Firebase Cloud Messaging [60] is a powerful tool that enables developers to send notifications and messages to users across various platforms. As part of Google's Firebase platform, it offers a comprehensive and flexible messaging solution that can be integrated into apps to enhance user engagement and communication. Figure 4.8 shows its official logo.

Figure 4.8: Firebase Messaging logo

It allows both developers and users to send notifications in two different ways: campaigns and messages. Campaigns are typically used by developers to inform users about events or promotions that might interest them, appearing as alerts in the device's notification bar. On the other hand, messages can carry a payload of data that is processed by the app in the background, allowing to create functionalities such as real-time chat or synchronization of app content without requiring direct user interaction.

Thanks to Firebase Cloud Messaging, it is possible to target messages to specific user segments based on various criteria, such as user behavior, demographics or preferences. It supports message scheduling and topic-based subscriptions, giving developers the ability to deliver the right content at the right time to their users.

4.4.5 Cloud Functions

Firebase Cloud Functions [58] is a powerful serverless tool that allows developers to run backend code in response to events triggered by Firebase features and HTTPS requests. They enable the creation of scalable microservices that automatically respond to database changes or authentication events, without the need to manage any infrastructure. Figure 4.9 shows its official logo.

Using Firebase Cloud Functions, the code is executed in a secure, managed environment, meaning that developers do not have to worry about provisioning servers or managing scaling. This allows them to focus on writing code that enhances the app's functionalities. Since functions are triggered by events, they run only when needed, making them cost-efficient.

Firebase Cloud Functions are tightly integrated with other Firebase services, such as Firestore and Firebase Authentication, which allows an ideal interaction across different parts of the app. Additionally, since the code runs in Google's Cloud infrastructure, it benefits from the same security, speed and reliability that other Google services have.

Figure 4.9: Firebase Cloud Functions logo

These functions can be written in JavaScript, TypeScript or Python. For this project, Javascript was used and functions were deployed via the Firebase CLI which allows to push updates or create new functions with minimal effort. The usage of this kind of feature is justified by the fact that it was not possible to use the provided Messaging v1 API through an HTTP library.

Chapter 5

Development

In this chapter, we focus on the development process of the application, emphasising the architectural choices and the specific libraries used to implement the features outlined in Chapter 3. We explore the reasoning behind these choices, including the advantages they bring in terms of maintainability, scalability and adaptability. Additionally, we discuss about the accessibility and the improvements done to allow other people to better use the application. The chapter concludes with the testing methodologies applied to ensure robustness of the application and a detailed explanation about the real environment testing.

5.1 Comparison of different architectures

The search for the most suitable structural pattern for our application involved the exploration of various options, including MVC, MVP, MVVM and VIPER.

Below there is a description of each pattern [39].

5.1.1 MVC (Model-View-Controller)

Description

MVC is one of the oldest and most widely known architectural patterns [14] [16]. Figure 5.1 shows the composition of the design pattern.

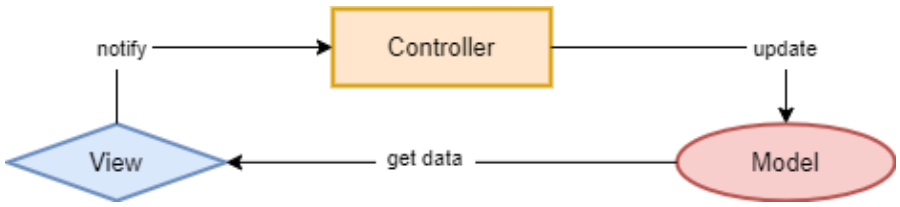


Figure 5.1: Model-View-Controller pattern [47]

It divides the application into three interconnected components:

- **Model:** it handles the business logic and interacts with the database or APIs to get and manipulate data. It also defines the rules for how the data can be manipulated, such as validation rules and constraints;
- **View:** it is responsible for displaying data to the user. It is the component that the user interacts with directly, such as buttons, text fields and images. It is also responsible for updating the user interface based on the data provided by the Model;
- **Controller:** it acts as the intermediary between Model and View. It handles user's inputs from the View and it updates the Model.

The data flow using this pattern works as follows:

1. **User Interaction:** the user interacts with the View;
2. **View to Controller:** the View notify the user's interaction to the Controller;
3. **Controller to Model:** the Controller processes the input and updates the Model;
4. **Model to View:** the View displays the updated data retrieved from the model;

Advantages

- **Simplicity:** usable for starting small projects or rapid prototyping;
- **Reusability:** components can be reused from different applications;
- **Separation of Concerns:** clear separation between data (Model), UI (View) and logic (Controller).

Drawbacks

- **Tight coupling:** view and controller can become tightly coupled, making changes and refactoring difficult;
- **Limited scalability:** not ideal for large-scale applications due to its structure.

5.1.2 MVP (Model-View-Presenter)

Description

MVP is an evolution of the MVC pattern, designed to make the View more passive and remove the tight coupling between the View and Controller [14] [16]. Figure 5.2 shows the composition of the design pattern.

It consists of:

- **Model:** similarly like in MVC, responsible for the business logic and data management;
- **View:** similarly like in MVC, the view component represents the user interface of the application;

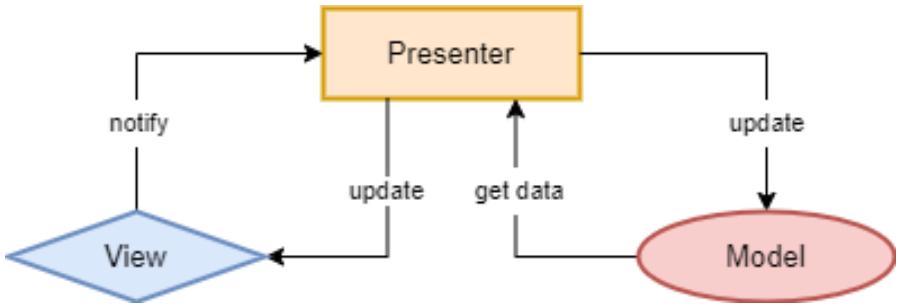


Figure 5.2: Model-View-Presenter pattern [47]

- **Presenter:** it is a component that connects the data (Model) and the user interface (View) together and manages the interaction between them. It receives input from the user, processes it and updates the view accordingly. It helps in separating the concerns of data and user interface.

The data flow using this pattern works as follows:

1. **User Interaction:** the user interacts with the View;
2. **View to Presenter:** the View sends user actions to the Presenter;
3. **Presenter to Model:** the Presenter processes the events and waits for updates from the Model;
4. **Model to Presenter:** the Model returns the changes to the Presenter;
5. **Presenter to View:** The Presenter updates the View with the new data.

It is important to underline how in the MVP pattern the Model and the View are independent and unaware of each other. The Model has no knowledge of the View and viceversa. The presenter acts as the intermediary, updating the Model in response to user's input and updating the View based on changes in the Model.

Advantages

- **Separation of Concerns:** better separation than MVC;
- **Testability:** easier to unit test as logic is separated into the Presenter;
- **Modularity:** code is more modular and easier to manage.

Drawbacks

- **Complexity:** increased complexity compared to MVC due to more responsibilities in the presenter.
- **Boilerplate Code:** often results in more boilerplate code, which can make the codebase harder to read and understand.

5.1.3 MVVM (Model-View-ViewModel)

Description

The Model-View-ViewModel (MVVM) architectural pattern is widely used in software development, especially in applications with rich user interfaces, like desktop and mobile apps. It provides a way to separate concerns within an application [14] [16].

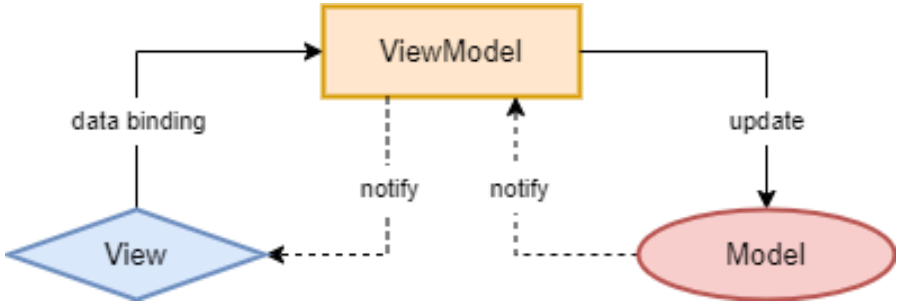


Figure 5.3: Model-View-ViewModel pattern [47]

It consists of:

- **Model:** similarly like in MVC and MVP, responsible for the business logic and data management;
- **View:** similarly like in MVC and MVP, the view component represents the user interface of the application;
- **ViewModel:** it serves as an intermediary between the Model and the View. It exposes the data and functionalities that the View needs while handling user's interactions and updating the Model as necessary.

The data flow using this pattern works as follows:

1. **User Interaction:** the user interacts with the View;
2. **View to ViewModel:** the View binds to the ViewModel, so that changes in the View trigger updates in the ViewModel;
3. **ViewModel to Model:** the ViewModel interacts with the Model to fetch or update data based on user actions;
4. **Model to ViewModel:** the Model sends updates to the ViewModel;
5. **ViewModel to View:** the ViewModel updates the View through data binding, automatically having the new data.

Advantages

- **Separation of Concerns:** this helps in managing the complexity of large applications organizing the code into distinct layers, each with a clear responsibility;

- **Maintainability:** clear separation of responsibilities makes the codebase easier to understand and maintain. Changes in the UI often require only modifications in the View or ViewModel, without affecting the business logic in the Model;
- **Testability:** isolating the business logic in the ViewModel, MVVM makes it easier to write unit tests. Since the ViewModel is independent of the UI, it can be tested without the need for a user interface, leading to more reliable and maintainable code;
- **Scalability:** it is well-suited for large applications with complex requirements.

Drawbacks

- **Complexity:** it introduces additional layers, which may not be necessary for simple applications. This could result in a lot of boilerplate code.

What differentiate MVVM from other patterns like MVC or MVP is its strict rules for component relationships. For example, the View has a reference to the ViewModel and the ViewModel has a reference to the Model, but it does not hold viceversa.

5.1.4 VIPER (View-Interactor-Presenter-Entity-Router)

Description

VIPER is a more complex architecture pattern that further separates concerns by introducing additional layers [14] [83].

It consists of:

- **View:** it displays data and interacts with the user;
- **Interactor:** it contains the business logic of the use case;
- **Presenter:** it contains view logic both for preparing content to display (by receiving it from the Interactor) and for reacting to user inputs (by requesting new data from the Interactor). It handles also the interactions with the Router for navigation;
- **Entity:** it represents the data models that the Interactor works with;
- **Router:** it manages navigation and handles the flow of screens.

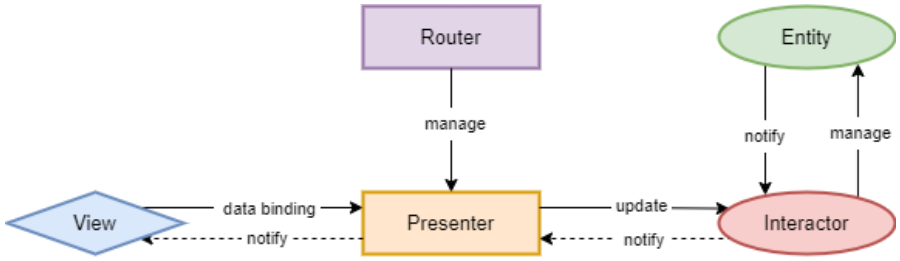


Figure 5.4: View-Interactor-Presenter-Entity-Router pattern [47]

The data flow using this pattern works as follows:

1. **User Interaction:** the user interacts with the View;
2. **View to Presenter:** the View sends user input or events to the Presenter;
3. **Presenter to Interactor:** the Presenter requests business logic or data operations from the Interactor;
4. **Interactor to Entity:** the Interactor queries or updates the Entity;
5. **Entity to Interactor:** the Entity provides updates to the Interactor;
6. **Interactor to Presenter:** the Interactor sends results back to the Presenter;
7. **Presenter to View:** the Presenter updates the View with new data or state;
8. **Router:** the Router intervene in navigation between different Views or modules if required.

Advantages

- **Separation of Concerns:** this helps in managing the complexity of large applications by organizing the code into distinct layers, each with a clear responsibility;
- **Scalability:** it is well-suited for large applications with complex requirements;
- **Testability:** since each component has a well-defined role, it enhances testability due to separation of the layers. In fact, this modular approach leads to a more robust and maintainable codebase.

Drawbacks

- **Complexity:** it is more complex than other patterns, with many layers to manage. This could result in a lot of boilerplate code.

Upon reviewing different architectural patterns, we identified MVP and MVC as structural patterns that are well-suited for small to medium-scale applications. However, due to their limitations in scalability, they were not considered for our needs. On the other hand, VIPER offers a robust structure, since it is considered an application of Clean Architecture [83], but it is used mostly for iOS development. In addition, the MVVM pattern combined with Clean Architecture is widely used in Android development [34].

Since one of the most important goals is having a high code quality, we had to find a compromise for what concerns architecture complexity without losing the stricly adhesion to the SOLID principles [82]. For this reason, we decided to adopt what we can consider a reduced version of the VIPER Architecture, that is similar to a MVVM with the Clean Architecture concept alongside the *BLoC* (Business Logic Component) pattern. By doing so, we declined the responsibilities typically held by the ViewModel into the *BLoC* state manager. Differently from Android, where *ObservableField* and *LiveData* are used, Flutter does not have built-in data binding, but it can be simulated by using streams and *BLoC* listeners [43].

In the following sections we describe more in detail about Clean Architecture and *BLoC*.

5.2 Clean Architecture

Clean Architecture [26] [50] is a software design principle that aims to organize and structure code in a manner that makes it maintainable, scalable and adaptable over time. It is widely used in the Android environment, but it is grown in popularity over the years. The primary goal of Clean Architecture is to achieve a high degree of separation of concerns, where different aspects of the application are isolated from each other, leading to a system that is easier to modify, extend and refactor. This design approach also emphasizes the independence of the core business logic from external entities, such as frameworks, user interfaces, databases or other third-party services.

5.2.1 Goals

The goals of Clean Architecture can be summarized as follows:

- **Independency from frameworks**: ensuring that the core business logic is not tightly coupled to any specific framework, software becomes more portable and adaptable to changes in the underlying technology. This allows us to swap or upgrade frameworks with minimal impact on the core application;
- **Independency from UI**: this makes the software more reusable and adaptable to the changes of UI;
- **Independency from database**: by decoupling the business logic from the database, software becomes more flexible and portable. This allows to have changes or upgrades to the database technology without requiring significant changes to the core application logic;
- **Independency from external agencies** (cloud providers or message brokers): it is designed to be independent of external systems, such as cloud providers or messaging systems. This makes the software more resilient to changes in the underlying infrastructure, enabling easier integration with other new services or providers.

5.2.2 Characteristics

Clean Architecture is composed of several layers, each with its own specific responsibilities and dependencies. These layers are designed to be loosely coupled, with the most stable and important business logic residing at the center and the more mutable positioned on the outer edges like external frameworks and infrastructure [26] [50].

Figure 5.5 shows the layers of the clean architecture. The arrows represents the fact that the outer layer have a dependency from a less to a more abstract part. So, for example, the entities are more abstract than the use cases. This concept is based on the SOLID's Dependency inversion principle state.

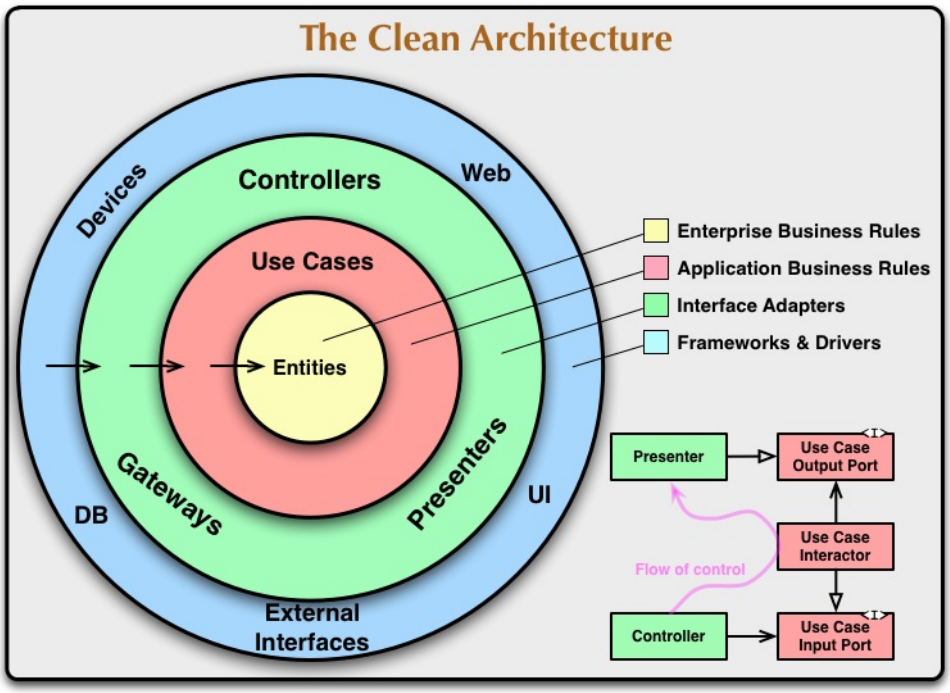


Figure 5.5: Clean Architecture

In particular we have:

- **Entities:** these are the core business objects that represent the main concepts of the application. They are independent from the framework, the data sources and the presentation layer and they define the properties and behaviors of the objects that the application has to manipulate;

- **Use Cases:** they contain specific application's business rules and interact with the entities. They are also independent from the framework, the data sources and the presentation layer and are typically organized around a specific operation or user's goal;

- **Interface Adapters:** it converts data from the use case into a format that can be used by the UI or other external layers. It is dependent on the Use Cases and Entities, but not on the framework or data sources;

- **Frameworks and Drivers:** the outermost layer, containing the frameworks, databases and UI elements.

By looking at the code of the *NetCare*'s application, it is clear that Use Cases have not been used. Still the goals of this approach were achieved. In fact, as also underlined by the inventor Robert C. Martin [50], it is possible to adapt the idea based on the needs of the project, since it is a design principle and it is very flexible in the interpretation.

A more detailed description of the typical project organization in a Clean Architecture is the following:

- **Data layer**: it is responsible for managing data access and storage. It includes code that interacts with external data sources, such as databases, APIs or other persistence mechanisms. In particular, it handles interactions with *Firebase Firestore*, *Firebase Storage* and other data sources, abstracting away the complexities of data access from the rest of the application;

- **Domain layer**: it contains the core business logic of the application. It defines the key entities, rules and operations that control the application's behavior. It is important to underline that this layer is independent of any specific technology or framework, ensuring that the business logic remains stable and unchanged, even in presence of technology updates;

- **Presentation layer**: it is responsible for handling user's interactions and presenting data to the user. It interacts with the Domain Layer to retrieve the necessary data and applies business logic as needed. It also includes the BLoC (*Business Logic Component*) pattern, which helps manage the state of the UI in a predictable and testable manner.

A key concept in Clean Architecture is the Service Locator pattern [80], which has been implemented in this project using the GetIt library [70]. The Service Locator pattern helps to reduce the number of dependencies between different parts of the application, leading to a more loosely coupled system. Thanks to the usage of GetIt, we can dynamically locate and provide instances of the necessary services, promoting flexibility and an easier maintenance.

The project is organized to ensure that each feature is independent one from the others, allowing for better modularity and easier testing. This modularity is achieved through the use of these folders:

- **features**: it contains all of the application's features, with each of them organized into the aforementioned layers (Data, Domain, Presentation);

- **core**: it contains all the shared features and utilities that are common across multiple features. This includes shared classes, utilities, services and widgets that can be reused throughout the application. The core folder helps to eliminate redundancy and ensures reusability across different parts of the application.

This organization aligns with the goals outlined in Section 5.2.1, ensuring that the application is maintainable, scalable and adaptable to future changes. For example, if the decision is switching from Firebase to another authentication provider, the architecture allows this to be done with minimal impact on the rest of the application. This can be achieved creating a new class that implements the necessary methods defined in the abstract class, without affecting other parts of the system.

Overall speaking, Clean Architecture provides a robust framework for building software that is resilient to changes, it is easy to maintain and flexible to evolving requirements. In fact, structuring the code into distinct layers and following the principles of Clean Architecture can ensure that the application remains flexible, scalable over time and capable of adapting to new technologies, platforms and users' needs.

5.3 BLoC

BLoC stands for *Business Logic Component*. It is a very popular state management pattern in Flutter and it separates the business logic of the application from the presentation logic. This separation of concerns leads to more maintainable, testable and scalable code. Figure 5.6 shows the components of the pattern.

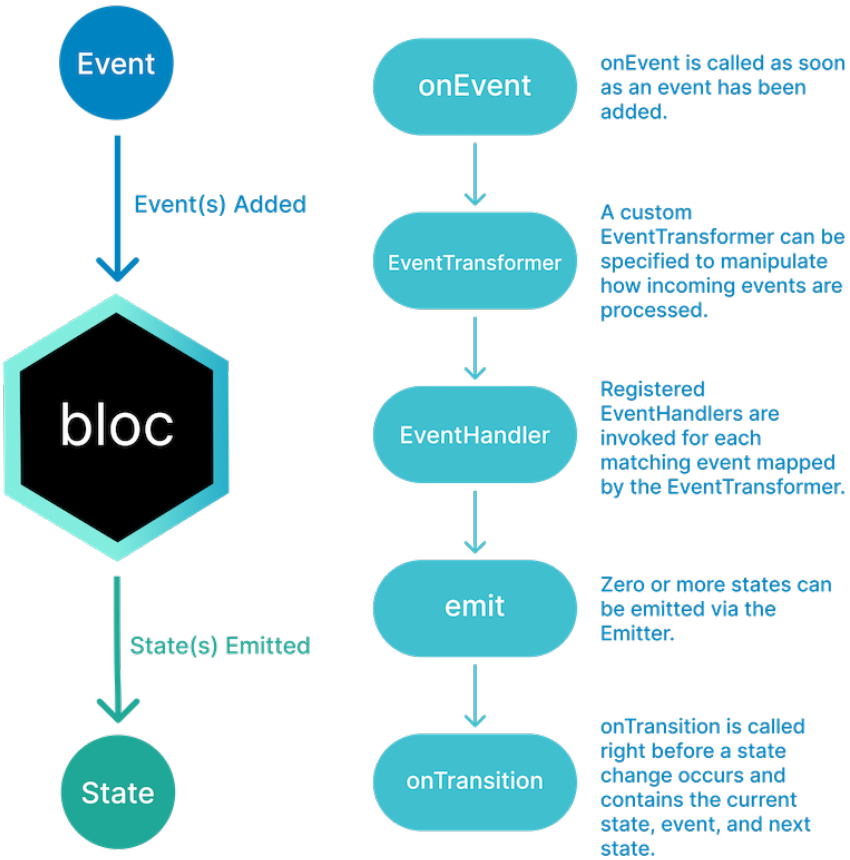


Figure 5.6: Bloc Pattern [44]

BLoC pattern is made up of 3 main concepts [42]:

- **States:** they are the outputs of the *BLoC*. After processing an event, the *BLoC* emits a new state, representing the current situation of the application. Each state should be a clear representation of the application's current condition;
- **Events:** they are actions or occurrences that trigger the *BLoC* to process some business logic;
- **BLoC**: this is where all the application's business logic resides. The *BLoC* is responsible for processing inputs (i.e, events) and producing outputs (i.e., states). It contains no direct references to UI elements, making it reusable across different features of the application.

The flow of the calls which involve the *BLoC* can be described as follows:

- **Adding event:** the UI sends an event to the *BLoC*, usually triggered by a user's action;
- **BLoC's event processing:** the *BLoC* receives the event and processes it. Depending on the event, the *BLoC* may perform some business logic;
- **State emission:** after processing the event, the *BLoC* emits a new state, representing the modified, but actual state of the application after the event's consumption;
- **UI Updates:** the UI listens to the *BLoC*'s state stream. When a new state is emitted, the UI rebuilds itself to reflect the new state.

It is important to underline that we did not move all UI state into *BLoC*s, but instead we used Material library's built-in controllers for certain elements like text input fields. This trade-off saved development time, but it deviates from best practice of fully separating state from UI. A future refactor could address this, achieving complete separation of state with its UI for all widgets.

Example

In order to better understand how the *BLoC* works, we propose an example extracted from the application. In particular, we focus our attention on the authentication feature in the case where the user would like to login.

So in Listing 5.1 and 5.2 there are the events and the states of the application for the authentication feature. Factory constructors are used to define different variants of a class. These different classes are represented as a separated class, but shares the same interface, which allows for pattern matching and the same sealed class behavior. We discuss more in detail about the freezed package in Section 5.4.

```
@freezed
class AuthState with _$AuthState {
  const factory AuthState.loading(String? text) =
  _loading;

  const factory AuthState.authenticated() =
  _authenticated;

  const factory AuthState.unAuthenticated() =
  _unAuthenticated;

  factory AuthState.initial() => const AuthState.unAuthenticated();

  const factory AuthState.otpError(String error) =
  _otpError;

  const factory AuthState.authError(String error) =
  _authError;
}
```

Listing 5.1: AuthState definition

```dart
@freezed
class AuthEvent with _$AuthEvent {
  const factory AuthEvent.signInRequest(String email, String password) =
  _signInRequest;

  const factory AuthEvent.signUpRequest(
    String email,
    String password,
    String name,
    String surname,
    bool isCaregiver,
    String otpCode) =
  _signUpRequest;

  const factory AuthEvent.signOutRequest() =
  _signOutRequest;
}
```

Listing 5.2: AuthEvent definition

In Listing 5.3, it is represented the *BLoC* and how it runs the business logic depending on which event is added from the UI.

```dart
class AuthBloc extends Bloc<AuthEvent, AuthState> {
  final AuthRepository _authRepository;

  AuthBloc({required AuthRepository authRepository})
  : _authRepository = authRepository,
  super(AuthState.initial()) {
    on<AuthEvent>(_onMapEventToState);
  }

  Future<void> _onMapEventToState(AuthEvent event, Emitter<AuthState> emit)
  ↪  async {
    await event.when(
      signInRequest: (email, password) =>
      _signInRequest(emit, email, password),
      signUpRequest: (email, password, name, surname, isCaregiver, otp) =>
      _signUpRequest(emit, email, password, name, surname, isCaregiver, otp),
      signOutRequest: () => _signOutRequest(emit),
    );
  }
```

Next page ...

Previous page ...

```
1
2   Future<void> _signInRequest(Emitter<AuthState> emit, String email, String
    ↪  password) async {
3     try {
4       emit(const AuthState.loading("Signing in ..."));
5       await _authRepository.signIn(email: email, password: password);
6       emit(const AuthState.authenticated());
7     } catch (e) {
8       // emit error state
9     }
10  }
11
12  Future<void> _signUpRequest(Emitter<AuthState> emit, String email, String
    ↪  password, String name, String surname, bool isCaregiver, String
    ↪  otpCode) async {
13    try {
14      emit(const AuthState.loading("Signing up ..."));
15      await _authRepository.signUp(
16        email: email,
17        password: password,
18        name: name,
19        surname: surname,
20        isCaregiver: isCaregiver,
21        otpCode: otpCode
22      );
23      emit(const AuthState.authenticated());
24    } catch (e) {
25      // emit error state
26    }
27  }
28
29  Future<void> _signOutRequest(Emitter<AuthState> emit) async {
30    emit(const AuthState.loading(null));
31    await _authRepository.signOut();
32    emit(const AuthState.unAuthenticated());
33  }
34 }
```

Listing 5.3: AuthBloc definition

Following what we explained above, let's take the call in the UI which represents the click on the login button. As we can notice from Listing 5.4, there is a listener which react if one of the states are emitted.

```dart
class LoginScreen extends StatefulWidget {
  // ...
}

class _LoginState extends State<LoginScreen> {
  // ...

  @override
  Widget build(BuildContext context) {

    // ...

    body: BlocConsumer<AuthBloc, AuthState>(
      listener: (context, state) {
        state.maybeWhen(
          authenticated: _authenticated,
          authError: (error) async => await _authError(context, error),
          orElse: _circularIndicator,
        );
      },
      builder: (context, state) {
        return state.maybeWhen(
          loading: (text) => _loading(text),
          unAuthenticated: _unauthenticated,
          orElse: _circularIndicator,
        );
      },
    ),

    //...

  }

  void _authenticated() {
    Navigator.pushReplacement(
      context, MaterialPageRoute(builder: (context) => const HomeScreen()));
  }

  Widget _unauthenticated() {

    // ...

    ElevatedButton(
      child: Text("Login"),
      onPressed: () {
        BlocProvider.of<AuthBloc>(context).add(
          AuthEvent.signInRequest(_emailController.text,
          ↪  _passwordController.text),
        );
      },
    ),
  }
}
```

Listing 5.4: SignInRequest event triggered in the UI

Focusing our attention on the action of signing in, these are the steps that involves the *BLoC* pattern:

1. at the initial stage, the state of the UI is *unauthenticated*, indicating that the user is not logged in;
2. when the user presses the button, the *signInRequest* event is added to the BLoC;
3. the *_mapEventToState*() function is called and in the pattern matching it is chosen the *signInRequest* function which is a lamda that passes the email and password as parameters and it calls the respective auxiliary function;
4. at the end of the auxiliary function a state is emitted both in successful and in error cases;
5. the UI listens to the state stream and it navigates to the next page or manages the eventual error.

Additionally, the *flutter_bloc* library provides also several widgets to facilitate the interaction between *BLoC*s and the UI [66]. In particular, there are these widgets available:

- **BlocProvider:** it is a widget that provides an instance of a *BLoC* and it ensures that it is available to the widget tree to its children. It can be thought as a dependency injection mechanism, making the *BLoC* accessible to any widget that needs it. Listing 5.5 shows an example of definition.

```
BlocProvider(
  create: (context) => MyBloc(),
  child: MyWidget(),
);
```

Listing 5.5: BlocProvider example

- **BlocBuilder:** it is a widget that rebuilds its UI in response to new states. It takes a *BLoC* and a builder function as parameters and it is called whenever the *BLoC* emits a new state, allowing the UI to react to changes in the state. Listing 5.6 shows an example of definition.

```
BlocBuilder<MyBloc, MyState>(
  builder: (context, state) {
    if(state is MyState1){
      return MyWidget1();
    } else if(state is MyState2) {
      return MyWidget2();
    }
    // ...
  },
);
```

Listing 5.6: BlocBuilder example

- **BlocListener:** it is a widget that listens for state changes and performs actions, but it does not rebuild the UI. It is useful for showing notifications, dialogs or performing navigation in response to state changes. It works similarly to BlocBuilder but does not affect the UI directly. Listing 5.7 shows an example of definition.

```
BlocListener<MyBloc, MyState>(
  builder: (context, state) {
    if(state is MyState1){
      Navigate.of(context).pop();
    } else if(state is MyState2) {
      showDialog(
        context: context,
        builder: (context) => MyDialog(),
      );
    }
    // ...
  },
  child: MyWidget(),
);
```

Listing 5.7: BlocListener example

- **BlocConsumer:** it is a widget that combines the functionality of BlocBuilder and BlocListener. In particular, it allows to both rebuild the UI and perform side effects in response to state changes. This enhances the code readability and reduce the widget nesting. Listing 5.8 shows an example of definition.

```
BlocConsumer<MyBloc, MyState>(
  listener: (context, state) {
      // ...
  },
  builder: (context, state) {
      // ...
  },
);
```

Listing 5.8: BlocConsumer example

- **MultiBloc:** it is a widget thet provides MultiBlocProvider and MultiBlocListener for scenarios where we need to provide or listen to multiple *BLoC*s simultaneously. This enhances the code readability and reduce the widget nesting. In the example they are combined but they can be used singularly. Listing 5.9 shows an example of definition.

```
MultiBlocProvider(
  providers: [
    BlocProvider<MyBloc1>(
      create: (context) => MyBloc1(),
    ),
    BlocProvider<MyBloc2>(
      create: (context) => MyBloc2(),
    ),
  ],
  child: MultiBlocListener(
    listeners: [
      BlocListener<MyBloc1, MyState1>(
        listener: (context, state) {
          // ...
        },
      ),
      BlocListener<MyBloc2, MyState2>(
        listener: (context, state) {
          // ...
        },
      ),
    ],
    child: MyWidget(),
  );
);
```

Listing 5.9: MultiBlocProvider and MultiBlocListener example

In conclusion, flutter_bloc library offers a powerful way to manage state in a Flutter application decoupling business logic from UI code, making it easier to manage state transitions in a predictable manner. The key widgets facilitate the flow of data and state management within a Flutter app, resulting in a more structured development process, improved maintainability and more readable code. Regarding code readability, it can be further enhanced replacing *if-then-else* statements with pattern matching and in the next section we describe what was used to implement this.

5.4 Freezed

When building Flutter apps, developers often need to handle data models, especially when working with APIs. A popular package created to adress and facilitate this work is freezed [69]. This package is a code generator for Dart that provides an immutable class generator. It significantly simplifies the creation of data classes and sealed classes and it is often used in combination with json_serializable [76] in order to automatically generate also $fromJson()$ and $toJson()$ methods which are used a lot for serialization and deserialization.

Advantages

- **Immutable data classes:** it generates immutable data classes, meaning once a class is instantiated, its fields cannot be modified. This immutability is important for ensuring data integrity and making it easier to reason about state changes in the application;
- **Union types and sealed classes:** it supports union types and sealed classes, which are not natively supported in Dart. This allows us to define different states or variations of a class and handle them with pattern matching (e.g., via when or maybeWhen methods), which is particularly useful in state management;
- **Built-in copyWith method:** it automatically generates a $copyWith()$ method for classes. This method is useful for creating modified copies of an object changing a subset of its fields;
- **Equality and hashCode:** it automatically overrides the == operator and $hashCode$ for classes.

Drawbacks

- **Increased build times:** since freezed relies on code generation, it can slightly increase build times, especially in large projects with many data classes;
- **Verbose generated code:** code generated by freezed can be verbose and difficult to manage, especially when debugging.

So taking the previous example in Listing 5.7, instead of writing *if-then-else* statements, we can simplify the syntax writing something which appears like the following shown in Listing 5.10:

```
BlocListener<MyBloc, MyState>(
  builder: (context, state) => state.maybeWhen(
      myState1: _functionMyState1,
      myState2: _functionMyState2,
      // ...
      orElse () => null;
  ),
  child: MyWidget(),
);
```

Listing 5.10: BlocListener freezed example

5.5 Flutter Quill

Flutter Quill [67] is a powerful rich text editor library for Flutter, designed to provide a flexible and complete WYSIWYG (What You See Is What You Get) text editing experience. It is built on top of the Quill.js, a popular JavaScript library for rich text editing, and it brings a similar level of featuring to Flutter applications. The library exposes the following main features and characteristics:

- **Rich text editing:** it supports a wide range of text formatting options such as bold, italic, underline, strikethrough and various heading levels. It also includes support for bulleted and numbered lists, block quotes and code blocks, enabling developers to create and manipulate richly formatted text easily;
- **Embedded media:** the library allows the embedding of images, videos and other media types directly within the text editor. This feature is particularly useful for applications that require multimedia content editing, such as blogs or note-taking apps;
- **Customizable toolbar:** it has a customizable toolbar that provides easy access to text formatting tools. Developers can modify this toolbar to fit the specific needs of their application, adding or removing buttons or even creating custom formatting options;
- **Extensibility and customization:** the library is highly extensible, allowing developers to create custom formats, embed their own widgets and override default behaviors. This makes Flutter Quill suitable for a wide range of use cases, from simple text editing to complex document management systems;
- **Collaborative editing:** Flutter Quill supports collaborative editing features. This allows multiple users to edit the same document simultaneously, with real-time updates being reflected across all instances of the editor.

Flutter Quill is based also on 5 main concepts:

- **Document:** it represents the entire content of the editor. It is a structured data model that contains a series of text and format operations. The *Document* is responsible for maintaining the state of the text editor, including all text content, formatting and embedded media. The *Document* can be serialized into a JSON format for storage, transmission or processing. This is useful for saving the editor's content to a database, sharing it across devices or integrating it with other systems;
- **Delta JSON:** it is the operational data format used by Quill to represent changes to a *Document*. It is an expressive format that describes the differences between document's states, allowing for efficient storage and transmission of text and its formatting. *Delta* is composed of a sequence of operations, where each of them represents an action like inserting text, deleting text or retaining certain text with optional formatting [54]. In particular with *Delta* we have the following:
  - **Insert Operation:** it adds new text or embeds media into the *Document*. For example, `{ "insert": "Hello" }` would insert the text *"Hello"*;
  - **Delete Operation:** it removes a specified number of characters from the *Document*. For instance, `{ "delete": 5 }` would delete 5 characters;
  - **Retain Operation:** it keeps a range of text unchanged, but can also apply or remove formatting. For example, `{ "retain": 5, "attributes": { "bold": true } }` would retain the first 5 characters and make them bold.
- **Attributes:** they can be specified on operations to apply or remove formatting. These are represented as key-value pairs in the *Delta* JSON, such as `{ "insert": "Hello", "attributes": { "bold": true } }`, which would insert *"Hello"* as bold text;

- **Chaining:** *Deltas* can be chained together to describe complex changes to a document. This chaining allows for robust versioning and real-time collaboration, where multiple users can edit the document simultaneously, with changes merged without any conflicts;
- **Transformations:** it supports operations like composing (merging) *Deltas* and transforming them (i.e., resolving changes from different sources). This is particularly useful in collaborative editing scenarios, ensuring that all users see a consistent and up-to-date version of the document.

In this project there are no real-time collaboration documents, transformations or attributes since the purpose of the shared notes and events wants to be simple. On the other side, it could be improved in the future.

5.6 Mobile design analysis

In this section, we accurately make a design analysis of all the screens of the *NetCare* application. This analysis has the aim to explore the overall *NetCare* application's design from its visual presentation to its user experience. Thanks to examination of each screen and component, we can obtain an initial evaluation on how effectively the application serves its intended purpose and how well it supports users.

The evaluation process begins with an in-depth look at the User Interface (UI) design, where visual appearence, layout and colours are inspected. These elements contribute to the application's first impression and have an important charge in guiding users with their interactions with the app. A well-designed UI enhances aesthetic appeal, but also simplifies navigation and improves overall usability.

In addition, we also jump into the User Experience (UX) design, which is central to the application's effectiveness. This involves assessing the navigation flow, the responsiveness of the design and how much intuitive are the interactive elements. In order to have a user-friendly experience, it is essential to ensure that the app meets the needs of its users, enabling them to complete tasks efficiently and with minimal efforts.

5.6.1 General design considerations

In this section we highlight the general design considerations about *NetCare* application.

Device orientation

The application was designed to be used only in portrait mode. The reason behind this choice is based on the fact that we wanted to minimize the vertical scroll. For example, while looking to lists of tips or events or to the extended information from smartphones, we have found that if we gave the possibility to have the landscape mode, the user would have to scroll more in order to see all the information and this would have ruined user experience.

Furthermore, human eye is more used to see in portrait mode when it comes to screens and we thought it was the best solution also for how it is built the application.

Colours

The background of the screen is chosen specifically to avoid any visual problem. This simplicity is not just an aesthetic choice, but a functional one. By choosing a plain

background, the design ensures that the primary interactive elements are clearly visible.

This is further enhanced by the color scheme, which consists in a shades drawing of blue and light blue. These colors are not chosen by chance. In fact, blue and light blue were picked for their associations with calmness and serenity, which contribute to stress reduction [12] [23]. In addition, light blue has been shown to increase subjective alertness and performance on attention-based tasks [3] which is coherent with what we are working for. The light blue wrap the screen with a sense of tranquility, creating an inviting atmosphere that relax users.

Content size

Every dimension is expressed in relative units which is Scale-independent pixels for texts, % of height and width of the device for the rest. Thanks to the sizer package in Flutter [81], we were able to resize buttons, spaces, texts, etc according to the height and width of the used device. In this way we guaranteed that our UI is responsive and works on different devices.

In particular, almost all the text widgets in the app were made scalable with respect to the user's settings on the device. In this way, the user can make the text bigger thanks to the accessibility features and apps that support dynamic type will adjust according to the preferred reading size.

Widgets' size were also checked through Flutter Inspector so that we could diagnose layout issues. Thanks to this tool we also guaranteed the minimum size for widgets (i.e., 44px for buttons).

Images

In order to optimize storage efficiency and enhance loading speeds, we chose the PNG format for our images. This decision was made over JPEG, as PNG allows for the preservation of transparency in icons and illustrations, which is important for maintaining the visual integrity of our design elements. Additionally, PNG offers lossless compression, ensuring that image quality remains intact even after repeated loading and saving.

For what concerns the copyright and licensing, we used only proprietary or AI-generated images. This approach reduces any potential legal concerns related to copyright infringement, as all images are either owned by us or created through AI tools. For the AI-generated images, it is sufficient to declare them through a watermark or any declaration which is valid as defined in the EU AI Act [29]. So, we are fully compliant with all intellectual property and AI's regulations.

Navigation

As discussed in Chapter 2, many users have reported difficulties navigating the application. To address these concerns, it is important to provide intuitive navigation options, such as a clear menu or a back button, allowing users to easily orientate themselves and understand their current location within the app. Ideally, users should be able to reach their desired content with minimal interactions [77]. To improve the user experience, we implemented a hamburger menu within the top bar, along with a title on the right side that clearly indicates the user's current screen. Additionally, when navigating to temporary screens, a back button is provided to facilitate a quick return to the previous screen. This approach ensures that users can easily find their way around the app, making navigation straightforward and efficient.

5.6.2 Login

At the first launch of the application, a notification permission prompt is displayed to the user. This appears as the very first interaction and it is a choice that underscores the importance of understanding the users' preferences before they even begin to explore the app.

Following this initial prompt, the login screen appears to the user. The design of this screen is based on simplicity. It is created a clean and minimalistic layout where every element is carefully positioned to ensure that the user can immediately identify the most important fields and buttons. Emphasizing the *"Signup now!"* with an underline reinforces the fact that it is an interactive element.

Finally, all user's interactions occur internally to what is called the comfort zone, which is the area where the user feels more comfortable while interacting with the mobile phone.

Figure 5.7 shows some screenshots about the signin feature.

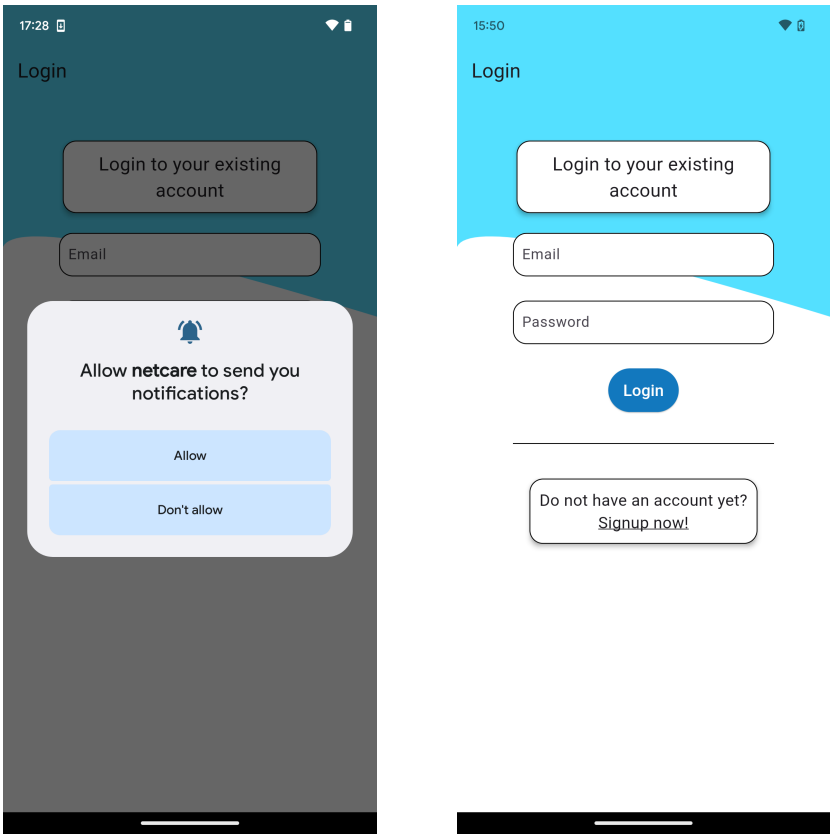


Figure 5.7: Notifications permission request and login screen

5.6.3 Signup

This screen appears after clicking to the underlined *"Signup now!"*. It represents a simple form in which it is requested to fill it with all the information. We intentionally minimized the number of fields to reduce user stress, as fewer fields result in a faster compilation time and this is better for the user.

From an aestethic point of view this screen does not present any particular differences with respect to the login screen.

Figure 5.8 shows some screenshots about the signup feature.

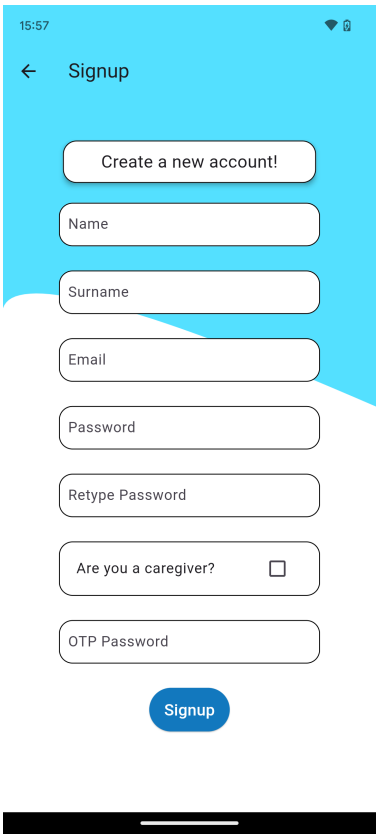


Figure 5.8: Signup screen

5.6.4 Home page

The *Home page* shows two cards that present daily tips, which are randomly selected from the database and changed every day. Each tip remains the same during all the day, encouraging users to interact with the application at least once daily.

The goal here is to create a connection between the user and the application, turning a simple daily tip into an interactive experience.

To maintain curiosity and engagement, we do not reveal the content of the cards upfront. Instead, we use a phrase that invites the user to click on it. This approach adds a touch of personality to the turtle character, making the interaction more engaging

for the user.

Figure 5.9 shows some screenshots about the home page feature.

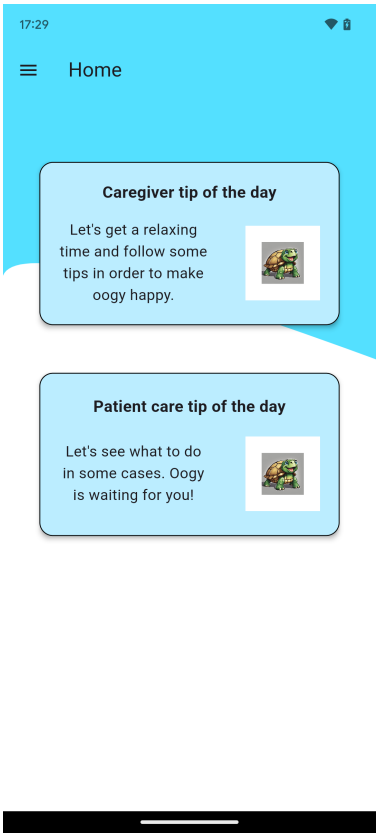


Figure 5.9: Home page screen

5.6.5 Coordination

The *Coordination* feature is made of a lot of screens. For this reason we divide the description into the following:

- Waiting Room;
- Family;
- Calendar;
- Pending Events;
- Activities.

Waiting Room

The *Waiting Room* is a part of the *NetCare* application where users can wait until they receive one or more requests to join a *room*. Also in this case we decided to mantain a clean widget setup with simple and effective visualization.

Figure 5.10 shows some screenshots about the waiting room in different states.

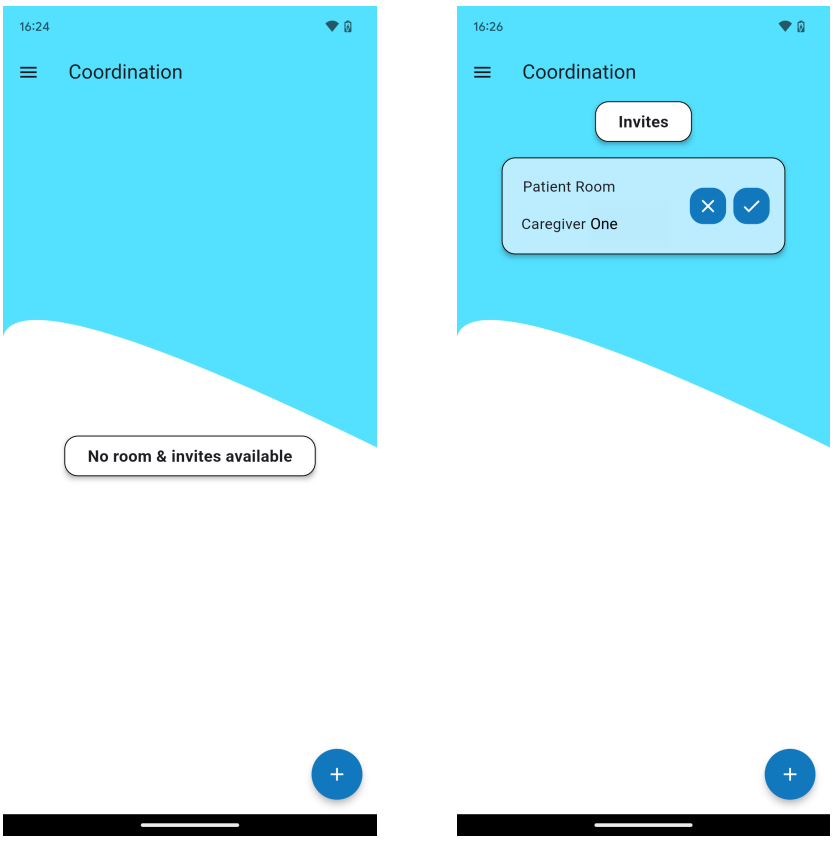


Figure 5.10: Different Waiting Room states

In addition, caregivers who wish to create a new *room* can do so by clicking on the floating button *"+"* located at the bottom right corner of the screen. Upon clicking this button, a creation dialog appears, allowing caregivers to easily set up a new *room*. The button's position is intentionally placed outside the comfort zone because users do not frequently create *rooms*. Since patients are not allowed to create *rooms*, the button for this function is not displayed to them.

Figure 5.11 shows a screenshots about the creation of a *room*.

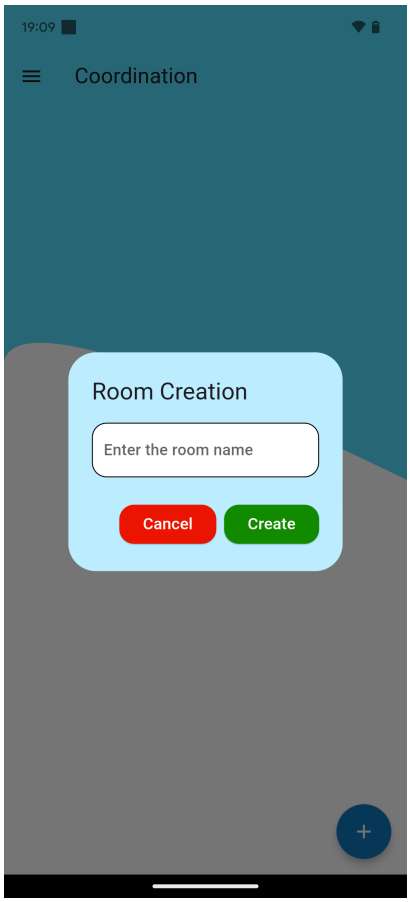


Figure 5.11: Room creation dialog

Room Menu

After entering in the *room*, the *Coordination menu* appears. The menu is very simple, with 3 different cards positioned in the center of the screen. The patient screen differs from the caregivers' one in that it displays "Activities" instead of "Pending Events" option.

It is important to note that there is a button at the top right of the screen, which varies depending on the type of user. Specifically, if the user is the caregiver host, he or she has the option to delete the room, which will affect all participants. On the other hand, if the user is not the host, they can leave the room without impacting the other members. Since it is a delicate action and it is used very few times, we decided to put it out to the comfort zone in order to avoid the user clicks by error. Clearly, a dialog to request confirmation is displayed in case it is triggered this action.

Figure 5.12, 5.13 and 5.14 show some screenshots about the room menu.

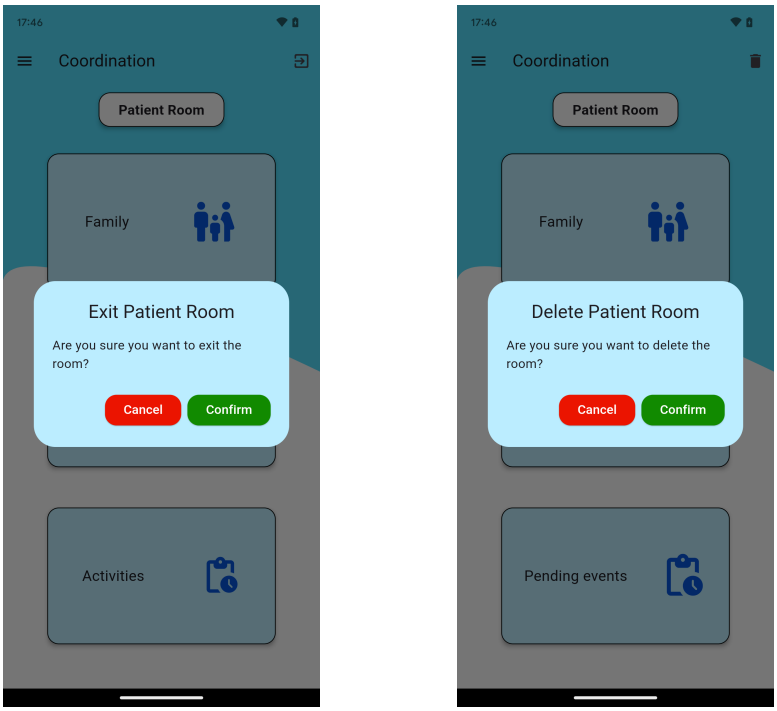


Figure 5.12: Exit/Delete room dialogs in coordination menu

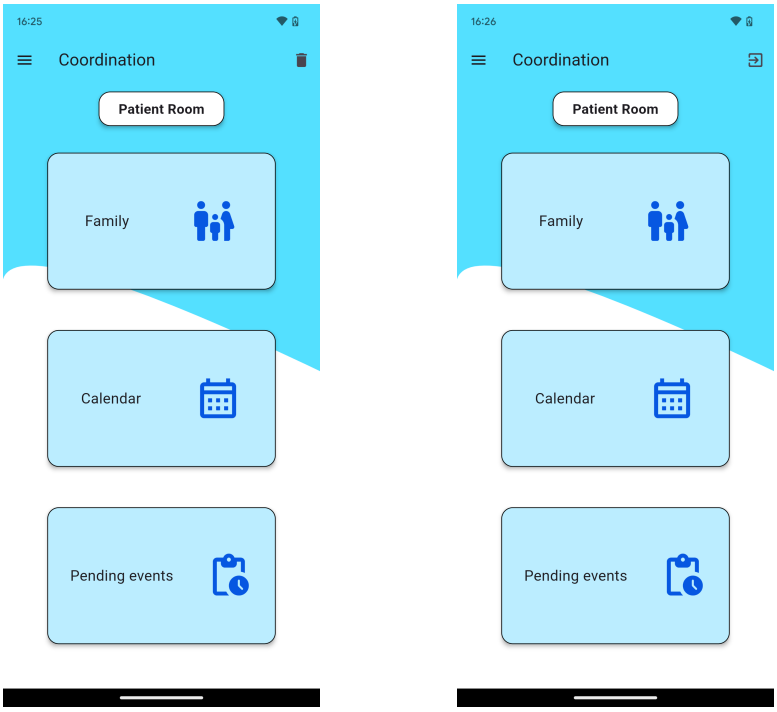


Figure 5.13: Host/Non-host caregiver coordination menu screen

Figure 5.14: Patient coordination menu screen

Family

The *Family* feature provides a summary of the participants in the room, helping users understand who is involved. This section uses cards to present key information about each participant.

The host has the option to add new participants by clicking the "+" floating button in the bottom right corner, which opens a dialog box. This dialog includes an input field and a temporary list where the host can save email addresses for sending invitations. Emails can also be removed from the list if needed.

However, hosts and users usually access this section to view the participant list rather than to add new members, so adding people is usually less frequent action.

Figure 5.15 shows some screenshots about the room menu.

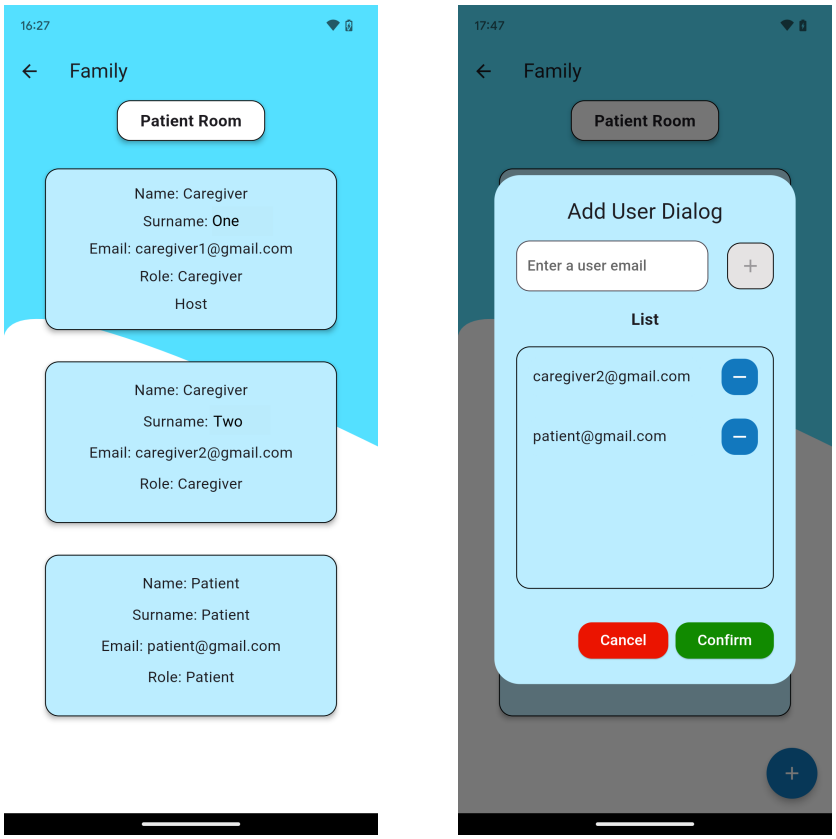


Figure 5.15: Non-host family screen and add users dialog

Calendar

The *Calendar* feature is the most essential part of the *NetCare* application. It displays in the foregroung a table calendar in the center of the screen, highlighting days with at least one event by showing dots. A maximum of two dots are shown per day, even if there are more than two events or activities. The user can also reduce the dimensions of the calendar with a 2-weeks or weekly representation.

Scrolling down, users can view a list of cards representing events or activities for the selected day. These cards are also horizontally scrollable, allowing users to easily navigate through them. This ensure the usage of gestures that are really appreciated by the users. When a card is scrolled, users can edit or delete the event or activity.

To add a new event or activity, users can click the "+" floating button in the bottom right corner, which makes them navigate to the event creation screen. Additionally, users can expand a card to view more detailed information about an occurrence. For convenience, an edit option is also available within the expanded information dialog, allowing users to navigate directly to the edit event screen. The add/edit event screen uses a card-based layout for fields. The key difference between the two screens is the additional option to set the recurrence of an occurrence during the creation process. Specifically, the picker uses red to indicate that the occurrence does not repeat on a given day, while green means that it does.

Furthermore, we can notice that all the main interactions with both the calendar and add/edit screens are in the comfort zone.

Figure 5.16, 5.17, 5.18 and 5.19 show some screenshots about the room menu.

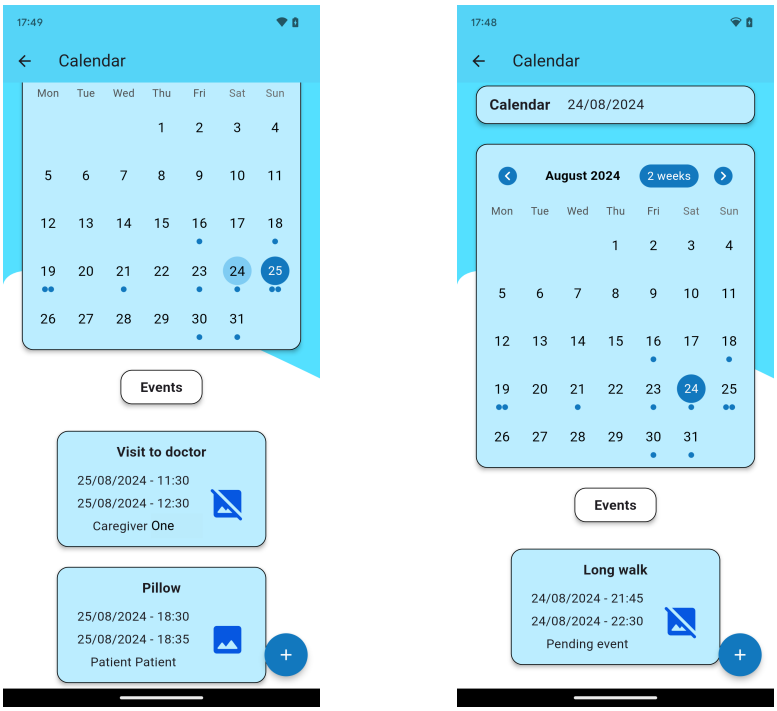


Figure 5.16: Event/activity assigned and pending event

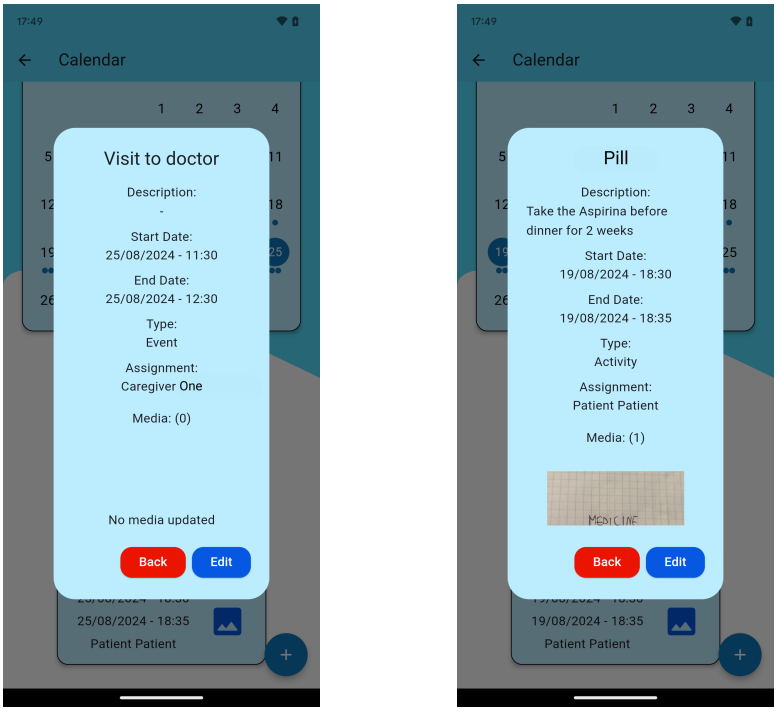


Figure 5.17: Extended information event/activity with/without images

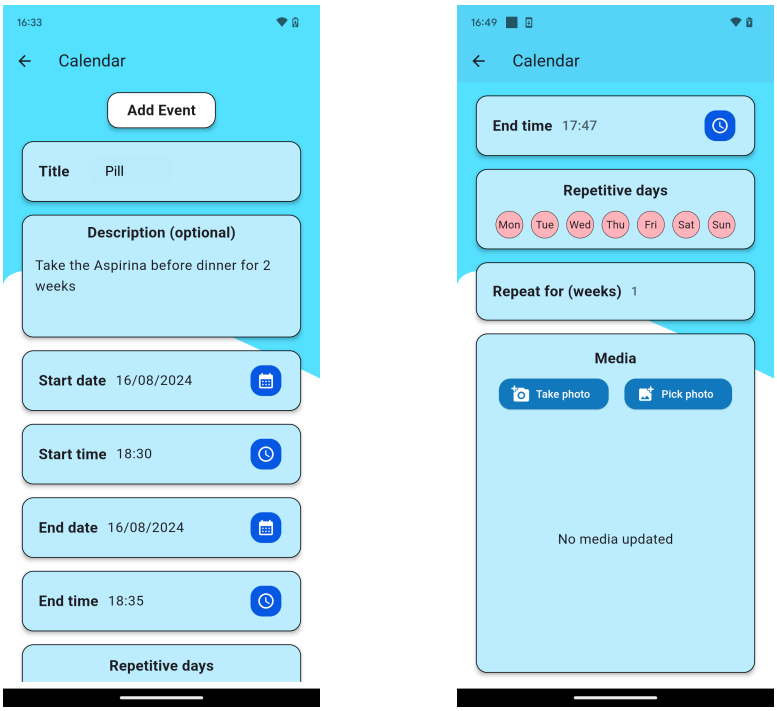


Figure 5.18: Add event screen

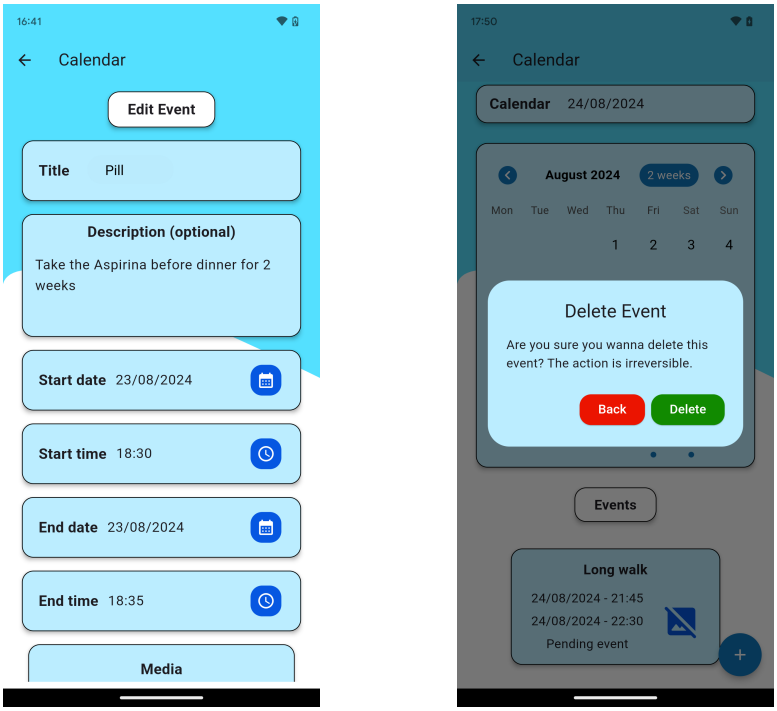


Figure 5.19: Edit event screen and delete event dialog

Pending Events

The *Pending Events* feature acts as a shortcut for caregivers, allowing them to see events that have not been assigned yet.

These events are displayed as cards, which can be expanded to reveal more details before making a decision. Caregivers can accept a pending event by clicking the "Accept" button in the detailed information dialog.

Figure 5.20 shows some screenshots about the Pending Events feature.

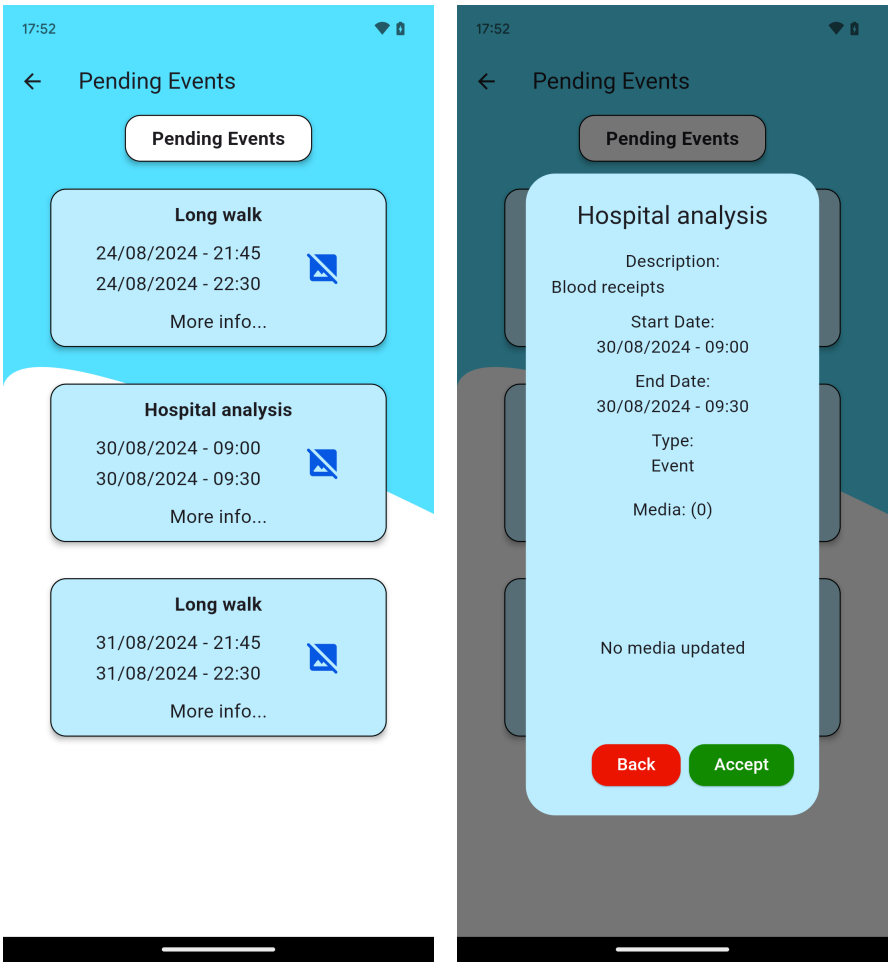


Figure 5.20: Pending events screen and extended information

Activities

The *Activities* feature is a functionality built for the patients in order to allow them to notify all the caregivers in the *room* about the completion of the activity. The layout and style of the screen is similar to the *Pending Activities* one. Furthermore, the patient can set the activity done by either clicking the checkmark on the card

or clicking the "Accept" button in the detailed information dialog. Moreover, before making their decision, users are requested to confirm their action.

Figure 5.21 shows some screenshots about the Activities feature.

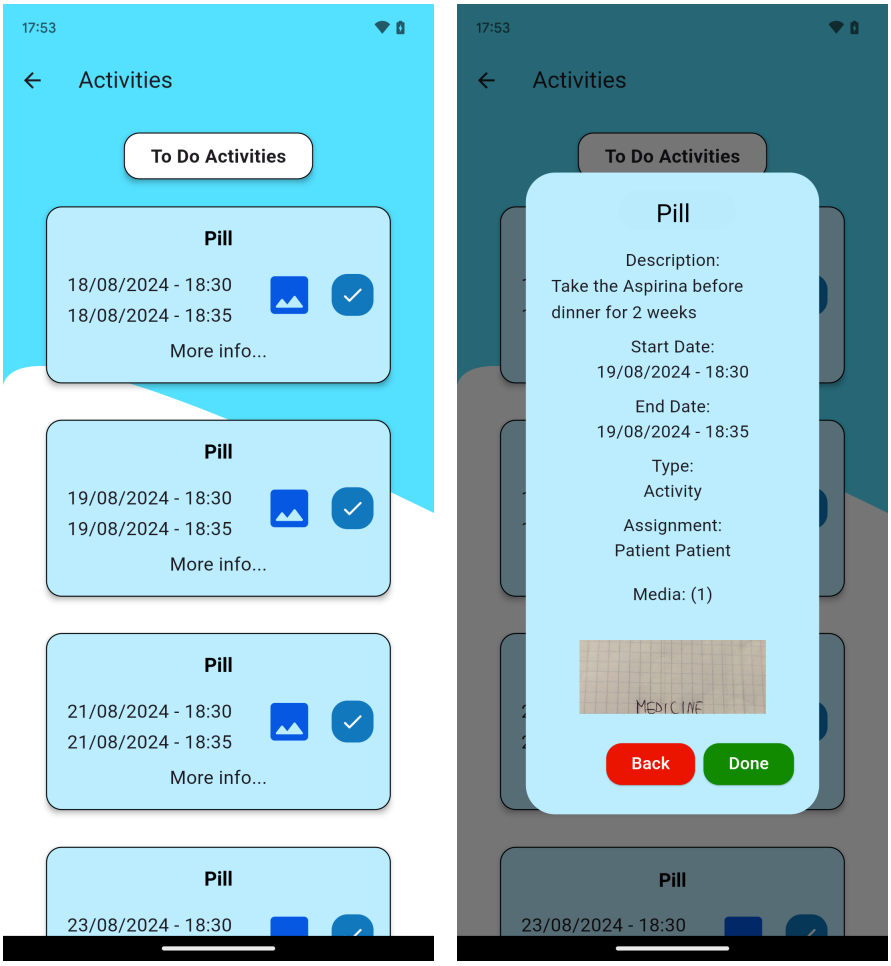


Figure 5.21: Pending activities screen and extended information

5.6.6 Notes

The *Notes* feature serves as a space for quickly jotting down some memos. For instance, users can save a quote from a doctor, the name of a drug or any other information that needs to be written down quickly. The layout is straightforward, having a filter text field at the top and a list of notes directly below it. Each note is represented as a card-based widget and it has similar appearance and functionalities with respect to those in the calendar feature.

In addition, when scrolling horizontally a note, users can access to a sharing functionality. When selected, a dialog appears, allowing users to scroll through the list of participants and choose whom to share the note with by selecting their tiles. Clearly, if the user is not in a *room*, the sharing functionality is disabled.

Figure 5.22 and 5.23 show some screenshots about the Notes feature.

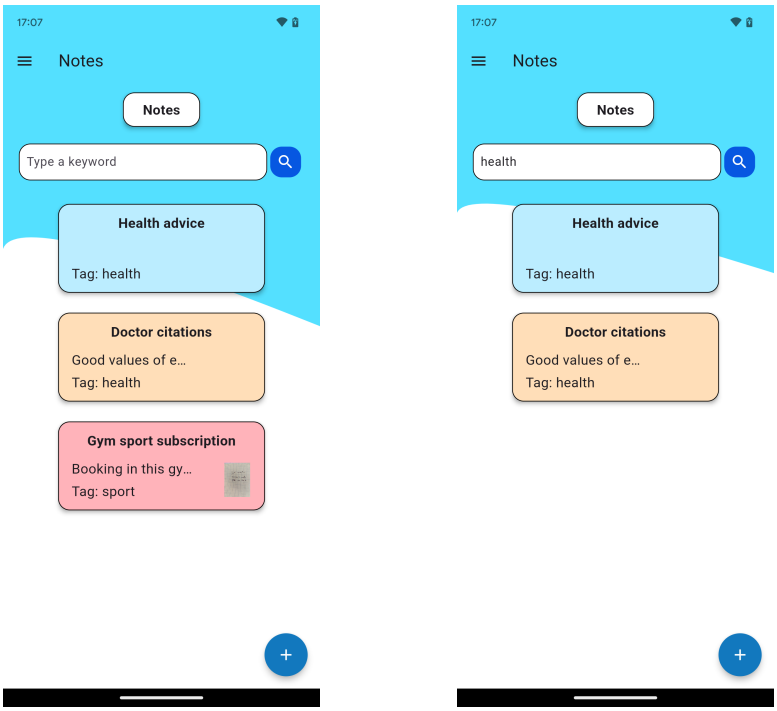


Figure 5.22: Notes screen and applied filter

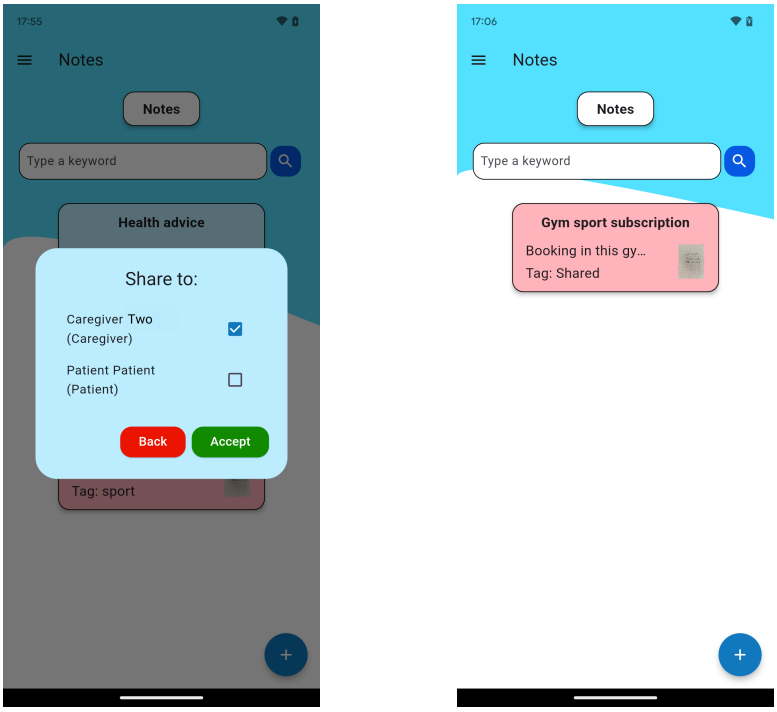


Figure 5.23: Sharing note dialog and shared note in another account

The add/edit screen also features card-based widgets, which include text fields for hashtags and titles, a color picker to change the card's color and a rich text field with formatting options for the content. If users click the back button, they are warned with a dialog about the possibility to lose the changes.

For what concerns images in notes, it is used the built-in method in the flutter_quill library [67] which offers additional functionalities, such as saving, zooming, resizing, copying etc. In addition, it allows to upload an image also using an existing link.

Furthermore, we can notice that all the main interactions with both the notes and add/edit screens are in the comfort zone.

Figure 5.24, 5.25, 5.26 and 5.27 show other screenshots about the Notes feature.

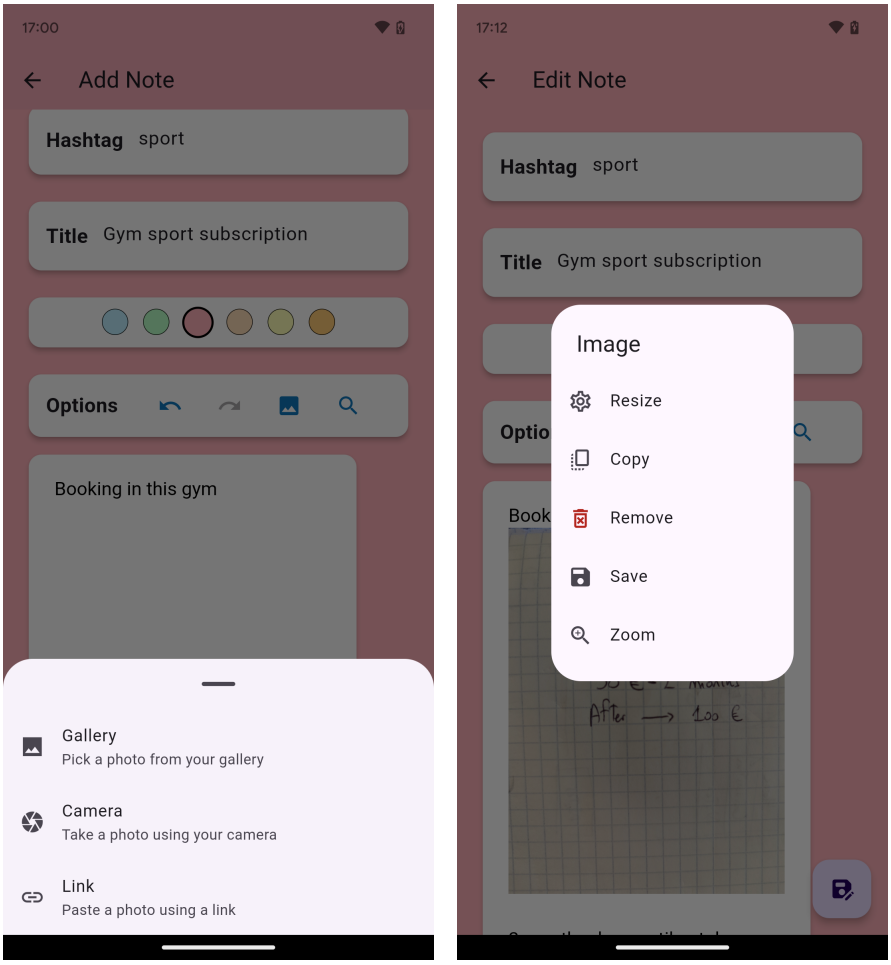


Figure 5.24: Picker image and modification in rich-text widget

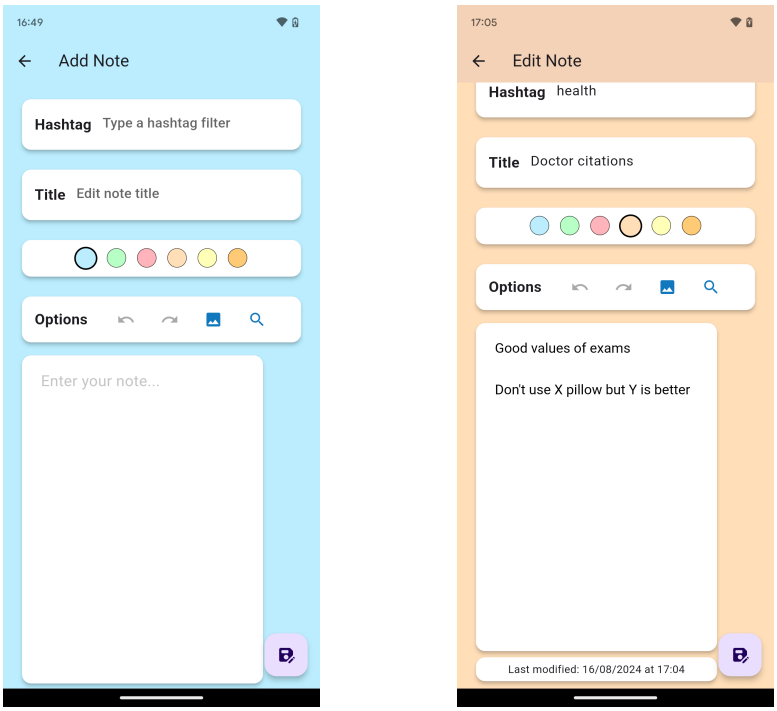


Figure 5.25: Create and edit note

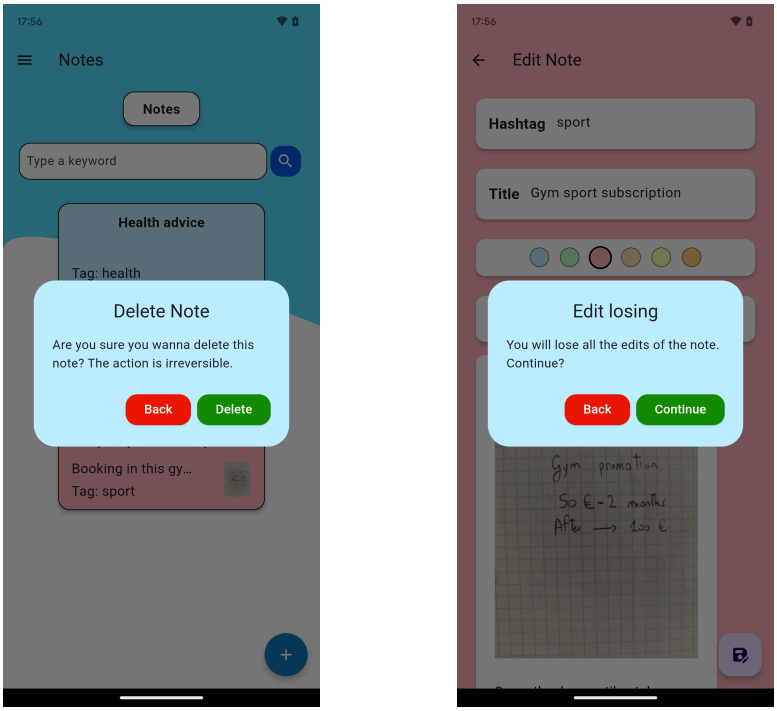


Figure 5.26: Delete note and back button warning

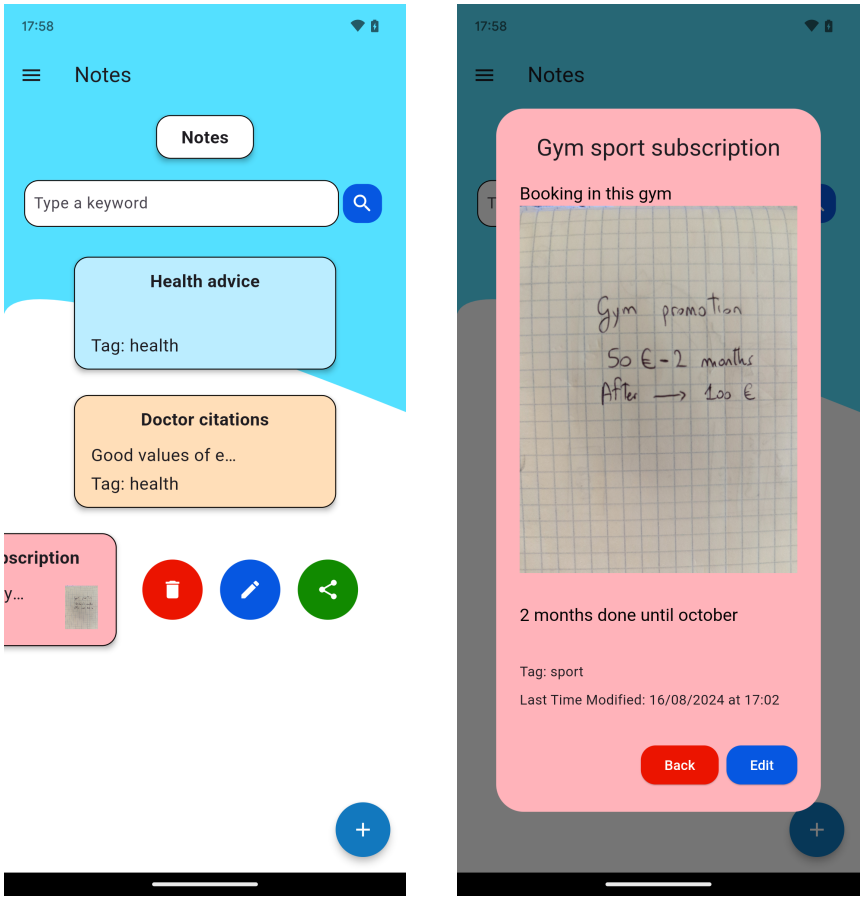


Figure 5.27: Notes slider and extended information

5.6.7 Tips

The *Tips* feature is designed to provide users, especially caregivers, with advice for better managing critical situations. The tips focus on both personal well-being and patient care, offering practical guidance for various scenarios with the primary goal of enhancing the user's overall health. The layout is card-based, with both caregivers and patient care sections sharing a similar design.

In the patient care section, users can filter tips based on the patient's phase of illness, ensuring that the advice is tailored to patient's current condition. For instance, tips for a patient in the second phase of dementia will differ from those for other phases. Meanwhile, in the caregivers' section, users can filter tips according to how much time they can dedicate to an exercise or activity. Additionally, both sections offer filters for various categories, allowing users to further refine their search.

The tips are presented as a list of cards and when a user clicks on a card, they can view an expanded version, similar to the extended views found in the home section.

Figure 5.28, 5.29 and 5.30 show some screenshots about the Tips feature.

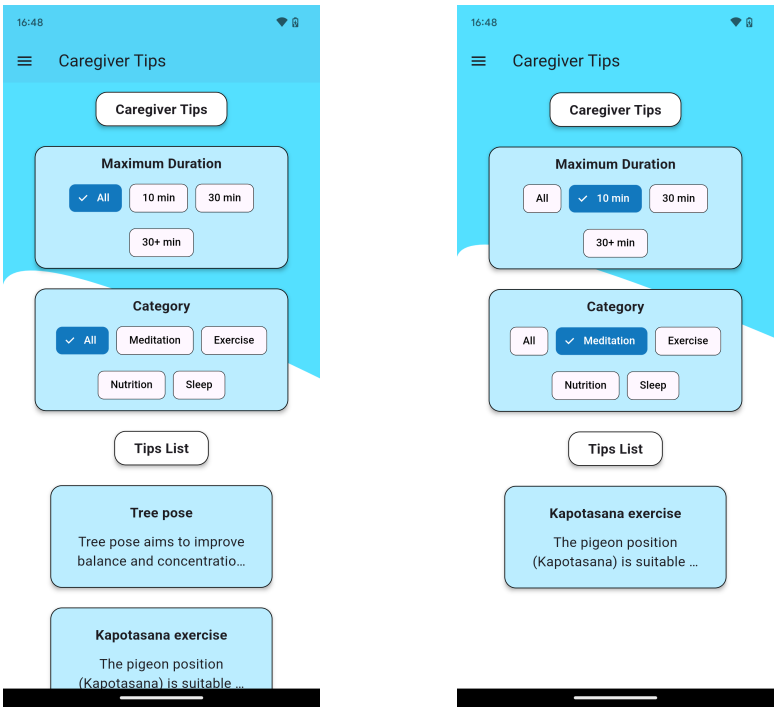


Figure 5.28: Caregiver tips not filtered and filtered

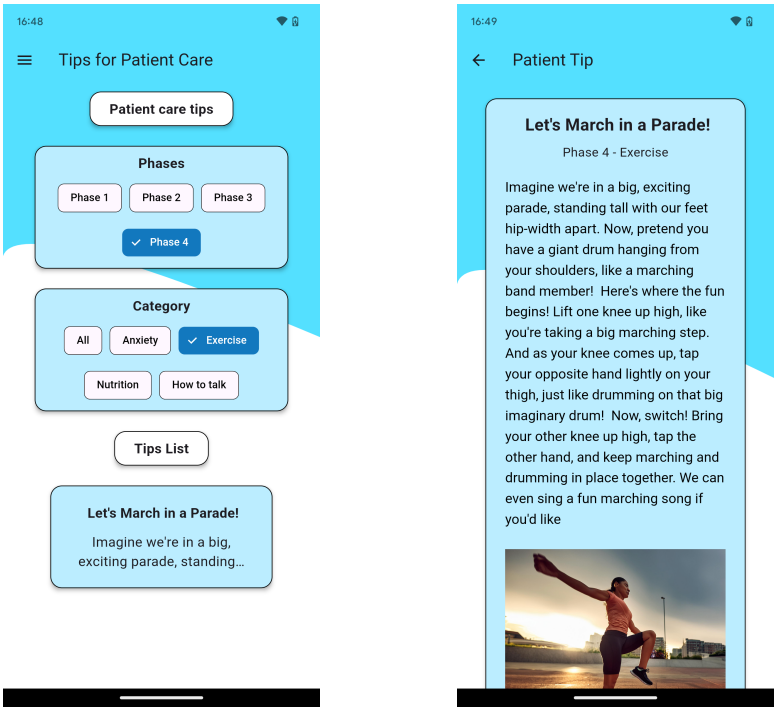


Figure 5.29: Patient tips filtered and extended information

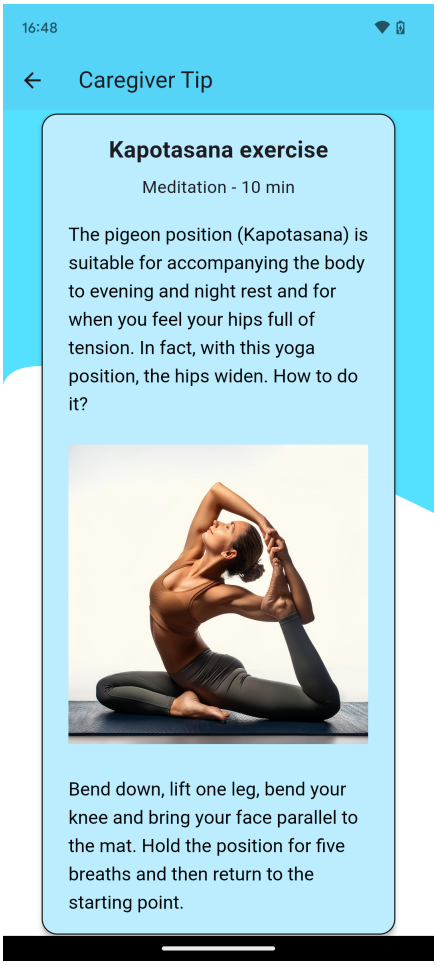


Figure 5.30: Caregiver extended information

5.6.8 Profile

This *Profile* screen acts as a personal resume, displaying user's profile photo and key information such as name, surname, and email. The user's profile photo and information are positioned at the top of the screen, allowing quick reading. Instead, the logout button is centrally located on the screen in the comfort zone for an easy access. If the user wants to perform a logout, a confirmation dialog box appears, ensuring that the action is intentional before proceeding.

Figure 5.31 shows some screenshots about the profile screen.

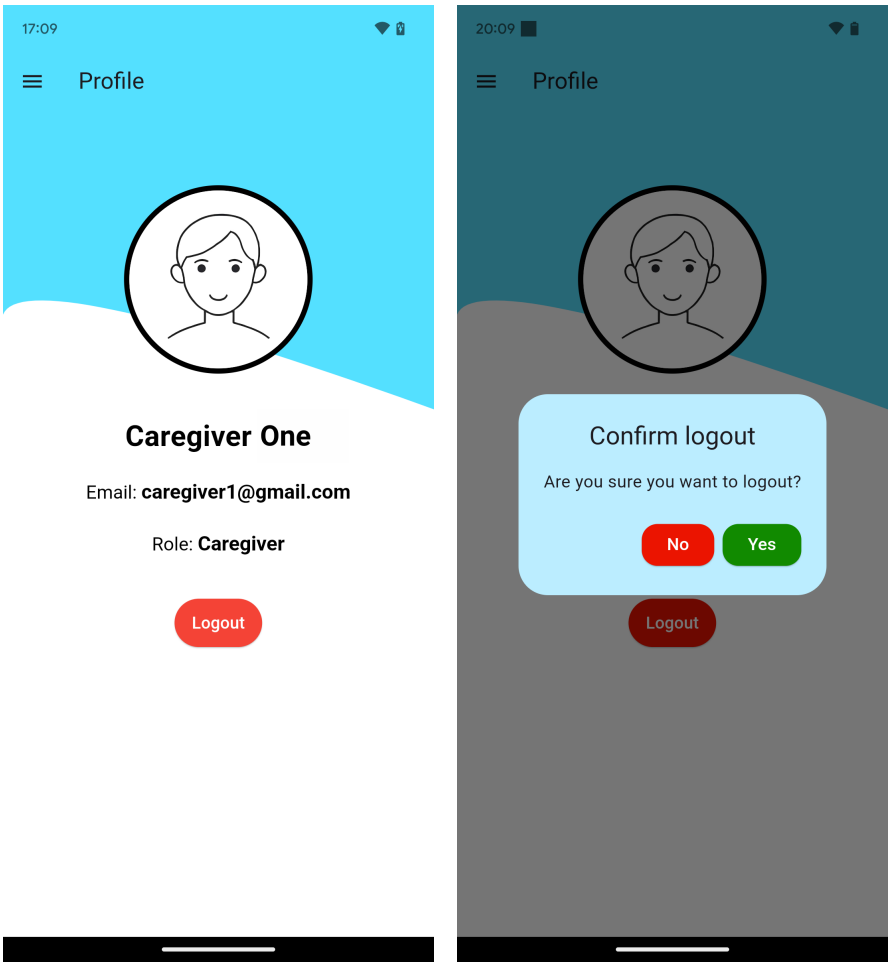


Figure 5.31: Profile screen and logout confirmation dialog

5.6.9 Helper dialog

This helper dialog was created to assist users with navigating the application and resolving any doubts they might have during its use. It is accessible through the hamburger menu and contains a simple slider in a dialog where a personified turtle guides users on how to use the less intuitive application's functionalities. This ensures users do not become frustrated when they are not sure on how to perform a specific action.

The user can either skip all the slides and go to the final one, making him saving a lot of time, or go to the next page by sliding or clicking the *"Next"* button.

As we can see from images, text is very coincisive and it was preferred to create an additional slide instead of putting all in one page.

Figure 5.32 and 5.33 show some screenshots about the helper dialog.

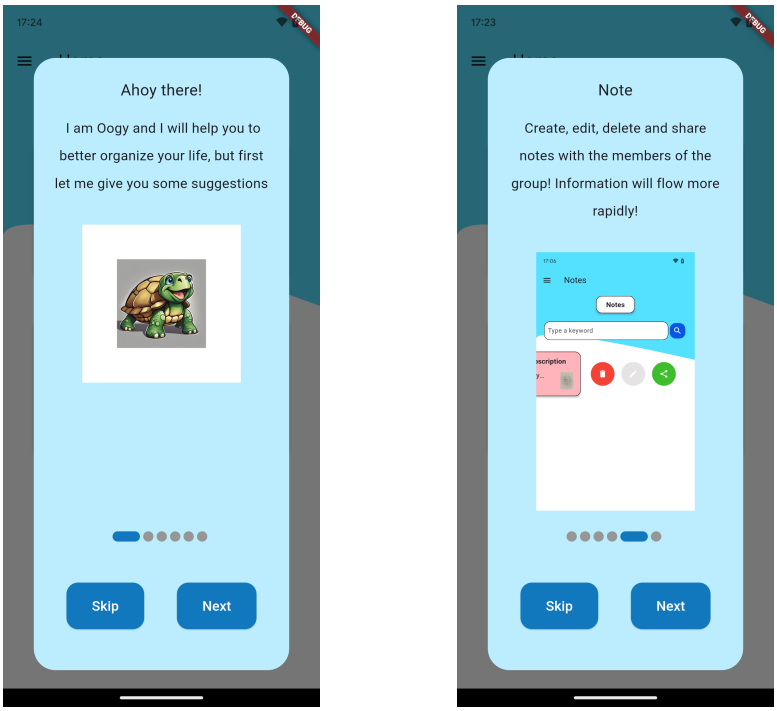


Figure 5.32: Helper slider at the beginning and center

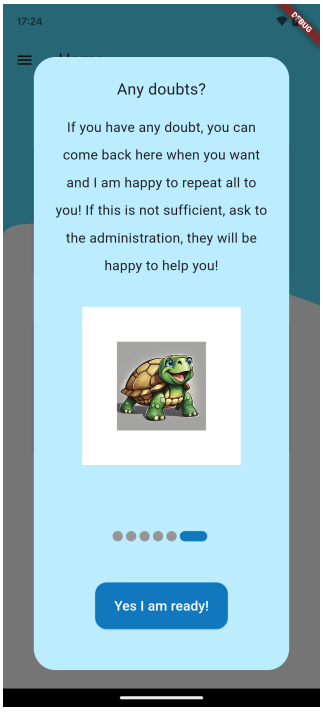


Figure 5.33: Helper slider at the end

5.6.10 Errors

The errors are presented as straightforward dialogs without any complex details, focusing on clear and direct communication. It is important to note that the application performs an internet connection check when the widget is built and so also when the user clicks *"Ok"* button, as maintaining an internet connection is essential for the app's functionality.

A unique case is the *One-Time Password* error. In this instance, the user is given two options: they can either retry the signup process or return directly to the sign-in page.

Figure 5.34 and 5.35 show some screenshots about the errors.

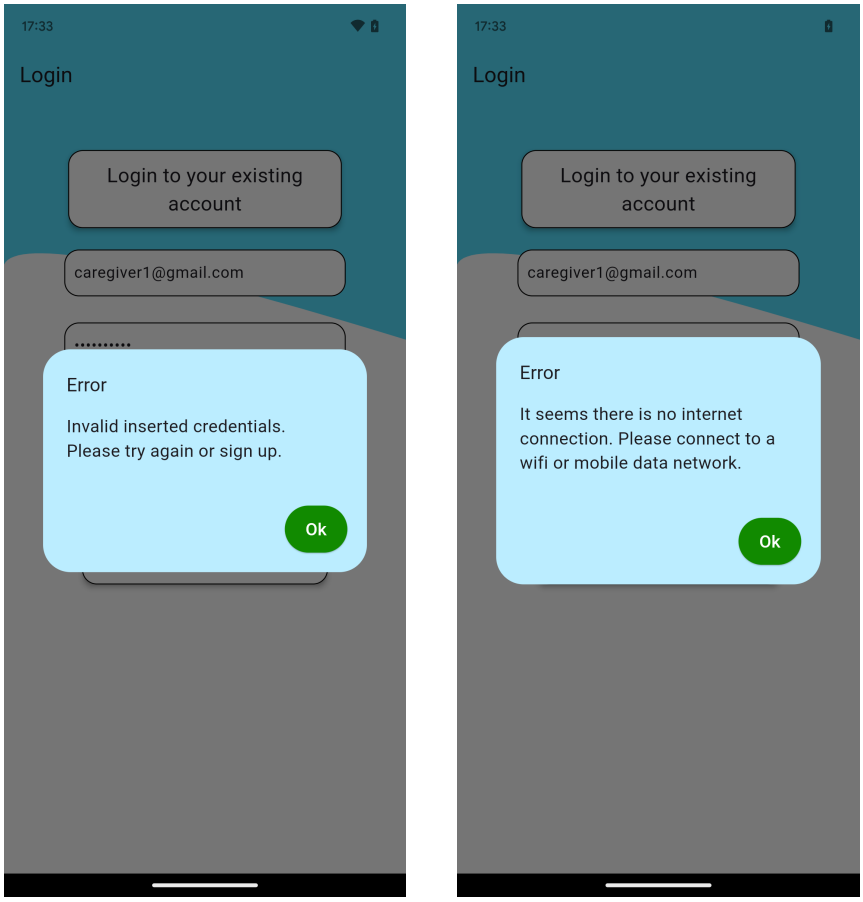


Figure 5.34: Invalid credentials and network errors

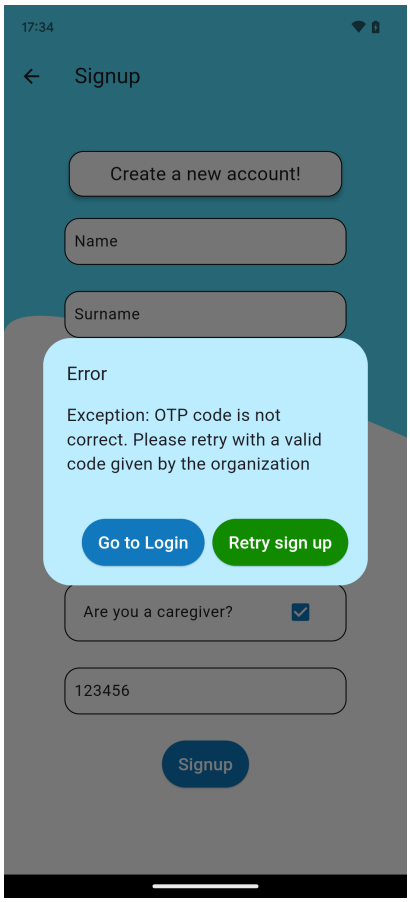


Figure 5.35: One-Time-Password error

5.7 Notifications

As we have already said in Section 4.4.5, it was not possible to use the Messaging v1 API through an HTTP library. It is strictly a security motivation in which each call have to be authorized thanks to a token. In order to speed up the development and avoid the need to build a backend specifically for this purpose, we used Firebase Cloud Functions. In particular, we chose First Generation Cloud Functions, as they provided all the necessary functionality without the need for additional features [51]. Basically we created an http call which:

1. takes the data passed from the application such as title, body and the sender's uuid;
2. retrieves the list of caregivers;
3. creates the payload;
4. sends the message for each caregiver in the list.

This approach allows us to bypass the complexities of backend infrastructure and we can directly create a function using the Firebase SDK APIs which let us create and send notifications in an fast and easy way.

5.8 Accessibility

Accessibility is an important aspect of mobile application design. In this context, accessibility takes a wide range of considerations to accommodate users with various impairments, including visual, auditory, motor, and cognitive disabilities. When developing the *NetCare* app, we adhered as much as possible to accessibility guidelines [77] [84]. However, while following these guidelines, we also considered the practicality and impact of different accessibility interventions in the context of our typical user base. For instance, it could be more reasonable that the patient having the Alzheimer could have also other types of impairments, but, on the other side, it may be uncommon for caregivers having visual impairments given the challenges they face in their role. For this reason, it could not be so much useful to add accessibility intervention in features where the typical user has a low probability to be impaired by definition. So it is important to find a trade-off between the intervention and what is is really needed. However, it is still important to make the application as accessible as possible for all users, ensuring broad accessibility to more people to use the application effectively.

In addition, the approach to accessibility in mobile applications must also consider future scalability and adaptability. As the user base of *NetCare* evolves and potentially includes a more different group of users, the app must be flexible enough to incorporate additional accessibility features if needed.

In fact, accessibility in mobile applications is not just a legal requirement in many jurisdictions, but it is also a moral need in order to create inclusive digital experiences that is inclusive for all users.

In the *NetCare* application, we focused on the following:

- zoom magnification;
- colour contrast;
- layout consistency;
- semantic.

Zoom magnification

In the group of people who needs accessibility features there are the ones having low-visual impariments. Most of the time they need a extension in widgets and text dimensions. Figures 5.36 and 5.37 illustrate the differences observed in some screens when the character and visualization dimensions in the accessibility section of the mobile phone settings are increased at the maximum.

Figure 5.36: Pending activities screen and extended information

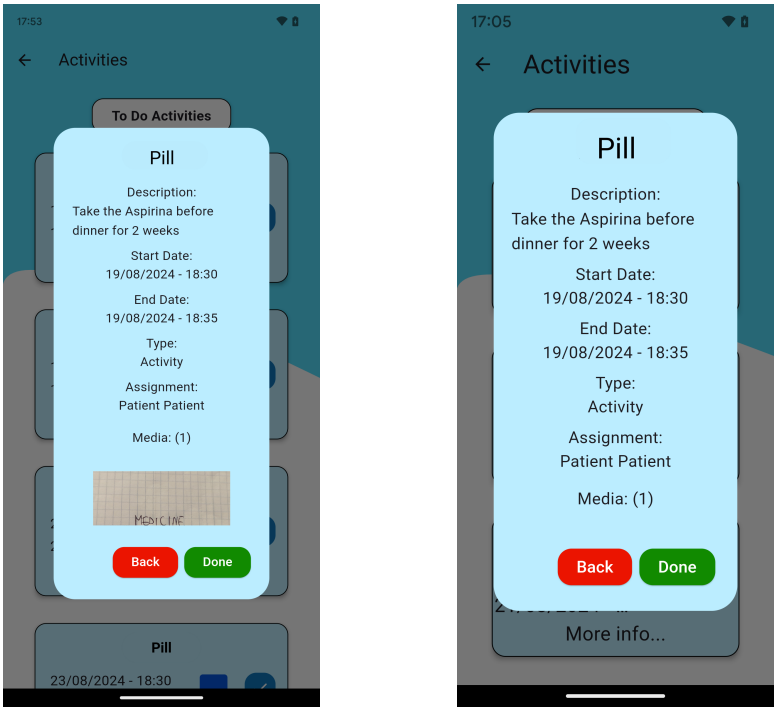


Figure 5.37: Default and bigger size text

It is clear that for the abitual user this is not a good user interface, but it can be readable for the ones who have difficulties in reading the information.

Colour contrast

Contrast is important for accessibility because it ensures that content is readable by everyone, including individuals with color blindness. High contrast between text and background helps users to distinguish text, images and other interface elements more easily, reducing eye struggling and improving readability. For those with color vision impairments, sufficient contrast ensures that important information is not lost or overlooked due to color distinctions that might not be apparent. This focus on accessibility benefits individuals with disabilities, but also improves the quality of the application for everyone [2]. Table 5.1 shows the level of contrast between colours in the application. It is important to emphasize that we only consider tests relevant to our specific test case. For instance, the combination of red text on a light blue background fails in standard text. Since we do not have any instance in the application, this combination can be excluded and we mark them with "-". The tests were done using the WebAIM contrast checker [52].

Background (HEX)	Foreground (HEX)	Small text	Big text	UI elements
White	Black	Pass	Pass	Pass
Turquoise	White	-	-	Fail
Turquoise	Black	-	-	Pass
Light blue	Black	Pass	Pass	Pass
Light blue	Green	-	-	Pass
Light blue	Red	-	-	Pass
Light blue	Dark blue	-	-	Pass
Light blue	Electric Blue	-	-	Pass
Dark blue	White	Pass	Pass	Pass
Electric Blue	White	-	-	Pass
Red	White	Pass	-	-
Green	White	Pass	-	-
Light grey	Black	-	-	Pass

Background (HEX)	Foreground (HEX)	Small text	Big text	UI elements
Light green	Black	-	-	Pass
Light red	Black	-	-	Pass
Light yellow	Black	-	-	Pass
Ocra yellow	Black	-	-	Pass
Orange peach	Black	-	-	Pass

Table 5.1: Colour contrast table

Hexadecimal code legend:

- White: *#FFFFFFFF*;
- Black: *#000000FF*;
- Turquoise: *#54E0FFFF*;
- Light blue: *#BBEEFFFF*;
- Dark blue: *#1078BEFF*;
- Electric Blue: *#000000FF*;
- Green: *#128A00FF*;
- Red: *#EB1400FF*;
- Light grey: *#E5E3E3FF*;
- Light green: *#B8FFC7FF*;
- Light red: *#FFB3BAFF*;
- Light yellow: *#FFFFB8FF*;
- Ocra yellow: *#FFCA75FF*;
- Light orange: *#FFDEB8FF*.

As we can see all tests were successfully passed, except for one. The combination turquoise-white fails in the UI elements test. Being flexible, since we encounter the combination in a very simple background graphics, it could be excluded because it is not so relevant and does not impact negatively the user experience. So, the application can be considered accessible for what concerns the colour contrast.

Gestures

Gestures have an important role in enhancing accessibility on mobile devices, making them easier to use for people with different needs.

They allow users to navigate interfaces, perform tasks and interact with content in a more intuitive and efficient way. For individuals with motor impairments, gestures like swiping, pinching or tapping can provide a more accessible alternative to traditional button presses, reducing the physical effort required.

Moreover, gestures enable screen readers to offer information about the context of the screen, making it easier for visually impaired users to understand and interact with mobile applications. In fact, by prioritizing gesture-based navigation, we can create more inclusive and user-friendly mobile experiences for these categories of people [2].

For what concerns the *NetCare* application, we added customized interactions in order to make some gestures easier. For example, the card slider is simplified allowing the selection of the action button by swiping up and down. This represents a huge improvement from an accesibility point of view. Listing 5.11 shows the creation of the custom interactions.

```
@override
Widget build(BuildContext context) {
  return SlidableAutoCloseBehavior(

    // ...

    child: Semantics(
      button: true,
      customSemanticsActions: {
        const CustomSemanticsAction(label: 'Delete'): () {
          _onDelete(context, index);
        }

        const CustomSemanticsAction(label: 'Edit'): () {
          _onEdit(context, index);
        },
      },
      child:

        //...
    ),
  );
}
```

Listing 5.11: Semantics in the buttons of the repetitive days picker

Layout consistency

Layout consistency is important for accessibility on mobile devices. In fact, it ensures a predictable and intuitive user experience for everyone, including those with disabilities.

A consistent layout helps users to navigate into the application, providing familiar visual patterns and predictable interactions and so reducing cognitive load [2]. By looking at the *NetCare* application, an example of layout consistency could be the extended information dialogs for notes and events/activities. In fact, for each dialog there are all the information related to the card, the back button and edit button respectively. This helps users to memorize the fact that they can edit something also when they are looking to the card's information. Another example could be the slider for the cards in the *Calendar* and *Notes* sections.

So, maintaining a consistent layout structure enhances *NetCare* application in both accessibility and user-friendliness not only for impaired people, but also for all the users in general.

Semantics

Flutter allows to make accessibility intervention in an easy way providing specific keywords and tools designed to enhance the semantic richness of an application.

In fact, Flutter's accessibility tools include the use of the *Semantics* widget [65], which allows developers to provide additional information about UI elements that might not be inherently obvious. This can include defining the role of buttons or describing complex visual elements and it enables screen readers to give the correct

information to the users. For instance, Listing 5.12 shows the usage of semantics in the buttons of the repetitive days picker. In particular, we force the screen reader to talk when a button is selected on not.

```
@override
Widget build(BuildContext context) {

  // ...

  return GestureDetector(
    onTap: () {
      onSelectedDay(index);
      WidgetsBinding.instance.addPostFrameCallback((_) {
        SemanticsService.announce(
          !isSelected ? '${Days.daysToInitial(day)} Selected' :
          ↪  '${Days.daysToInitial(day)} Deselected',
          TextDirection.ltr,
        );
      });
    },
    child: Semantics(
      button: true,
      checked: isSelected,
      excludeSemantics: true,
      child:
        // ...
    ),
  );
}
```

Listing 5.12: Semantics in the buttons of the repetitive days picker

We also used the *ExcludeSemantics* widget to enable users to skip sections of the application they may not be interested in. For instance, in the *Calendar* feature, we wrapped the *TableCalendar* widget with *ExcludeSemantics*, allowing users to skip the reading of the entire calendar. Since the primary goal on this screen is to quickly access the list of occurrences for a specific date, this approach enhances navigation efficiency for screen reader users. Instead of navigating through the entire calendar, users can simply change the date using a text field we added, making the process faster and more intuitive.

5.9 Test

In this section we discuss about application's testing from both a coding and real-life users' point of view.

5.9.1 Code testing

The testing phase is very important in application development in order to guarantee the integrity and correctness of the system. As we have already mentioned in Chapter

5.2.2, Clean Architecture ensures an easier way to test components thanks to its modularity.

In this project we used two libraries to make unit tests:

- **mockito:** it is a popular library for creating mock objects in unit tests. It is used to simulate the behavior of complex objects, such as services or dependencies, without requiring their real implementation [79];
- **bloc_test:** it is designed to make *BLoC* testing easier. It provides utilities to test the states and events of a *BLoC*, ensuring that the business logic behaves as expected. When used together with Mockito, bloc_test allows for comprehensive testing of *BLoC*s while mocking dependencies [45].

In particular, during development the focus was understanding the behavior of the emitted states, trying to identify edge cases and solve them accordingly. Listing 5.13 shows an example of how the *Caregivers Tips* feature was tested.

```
@GenerateMocks([CaregiverTipsRepository])
void main() {
  late CaregiverTipsBloc caregiverTipsBloc;
  late MockCaregiverTipsRepository mockCaregiverTipsRepository;

  setUp(() {
    mockCaregiverTipsRepository = MockCaregiverTipsRepository();
    caregiverTipsBloc = CaregiverTipsBloc(repository:
    ↪  mockCaregiverTipsRepository);
  });

  tearDown(() {
    caregiverTipsBloc.close();
  });

  group("CaregiverTipsBloc", () {

    // initialization ...

    blocTest<CaregiverTipsBloc, CaregiverTipsState>(
      "emits [loading, loaded] when getCaregiverTips is successful",
      build: () {
        when(mockCaregiverTipsRepository.getCaregiverTipsFiltered())
        .thenAnswer((_) async => caregiverTips);
        return caregiverTipsBloc;
      },
      act: (bloc) => bloc.add(const CaregiverTipsEvent.getCaregiverTips()),
      expect: () => [
        const CaregiverTipsState.loading(),
        CaregiverTipsState.loaded(caregiverTips),
      ],
    );
```

Next page ...

Previous page ...

```

    blocTest<CaregiverTipsBloc, CaregiverTipsState>(
      'emits [loading, error] when getCaregiverTips fails',
      // ...
    );
  });
}
```

Listing 5.13: Caregiver Tips unit test example

As we can see from Listing 5.13, the pattern used for testing [68] is the following:

- set up the mocks;
- tear down the mocks, useful when the *main*() function terminate the execution;
- create one or more groups of tests;
- create one or more single tests and use the *AAA* (Arrange, Act, Assert) pattern [40].

Thanks to this way of working, we ensured the correct *BLoC*'s emission of states before the development of the UI, guaranteeing the absence of problems for states emission during the UI's development.

5.9.2 Users testing

In order to see the effects of the *NetCare* application on users, it should be directly tested on a real environment in order also to refine its functionalities. In particular, the proposed framework works as follows:

1. find the testers group, which have to be at least of 1 Alzheimer's patient and 4 or more caregivers;
2. give access to an application's account to the users;
3. have a meeting after a month in which the caregivers give a feedback;
4. modify the application based on users' feedback;
5. iterate.

This approach should be applied to as many groups as possible to enhance the accuracy of the statistical analysis and avoid errors like the ones observed in some of the papers mentioned in Section 2.

Unfortunately, identifying this specific category of participants is challenging and, even if they are found, it is not sure that are going to entirely accept the framwork. Additionally, the timing of the testing phase was not ideal, since many potential participants were on holidays. Even if there were available groups, the number and the amount time they could dedicate to the application would be too low and this could make tests less significant. For this reason, we decided to subject the application

to a group of experts in order to have a deeper understanding of whether the work was good in quality and was able to have a meaningful psychological impact on users or not. During the meeting, we presented the *NetCare* application to the group of experts, who reviewed its features, functionalities and potential benefits. They provided valuable insights and feedback, expressing their satisfaction to what we achieved. They recognised the application's potential and how it could significantly improve the well-being of both Alzheimer's patients and their caregivers. We also discussed in a costructive way about additional features and functionalities that could increase the quality of the application and it will be discussed in Chapter 6.

In conclusion, their positive feedback confirms the quality of the work done and it highlights the need of testing it on people as much as possible.

Chapter 6

Conclusions and future works

In the previous sections, we detailed the work done for the *NetCare* prototype. Our process began with an extensive analysis of the current state of the art about informal caregiving with a particular focus on Alzheimer's patients. This research helped us to define the product properly and we presented the included and excluded features. Then, we had an overview on the technologies used for the project and described the development phase, emphasizing key structural patterns and design choices that characterised the application. Finally, we finished describing the testing phase, by focusing on the code testing, how to test the application in a real-world scenario and the feedback from experts to validate the application's effectiveness.

In the next sections, we present the limitations of our work and suggest future works emerged as possible extensions to the *NetCare* application. Finally, we conclude with some closing remarks.

6.1 Limitations

During the meeting with experts, they expressed a positive evaluation of the application. However, some limitations were identified in this study.

The most significant is the lack of testing in a real-world scenario, which means there is no quantitative data available. The reasons for this are explained in Section 5.9.2 and it is justified by considering *NetCare* as a initial step in a broader study. In addition, testing directly on the user should be very important in order to have the possibility to discuss directly with them and trying to improve the existing feature and create new ones based on their feedback.

The experts also highlighted a potential weakness in the application's patient-centric approach. Even if it is an application in which satisfying needs of caregiver and reduce their burden are the main goals, patients might feel subordinate or controlled. This could be a problem, in particular in the first stages of the disease. To address this, it is crucial to find a way to ensure patients do not feel this way. However, a real-world trial would be necessary to fully understand how patients at different stages might react to the application.

6.2 Future works

In this section we highlight the most promising extensions for the project.

As we have already mentioned in Section 3.5, NFC notifications, health documentation and GPS localization could be considered very important features. In fact, they could significantly enhance the organization of caregivers responsibilities by automatically notifying caregivers about a completed task, informing about updates of health documentation or warning whether the patient could potentially be lost or not. These could be a huge improvement, but, on the other side, it is important to show the compatibility with the European Union's GDPR [27].

Additional extension and improvements for the project emerged during the meeting with experts, which are the following:

- notifications;
- media;
- chatbot and live chat;
- emergency call.

6.2.1 Notifications

In additon to NFC notifications, a potential extension could involve enhancements for what concerns the notifications. In the actual version of the *NetCare* application, notifications are used for 2 main goals:

- pushing the users to use more the application, thanks to campaigns;
- informing the caregivers about the completion of an activity.

During the meeting with experts, they suggested other different notification's use cases in order to make the system more efficient in terms od communication. In particular, the application should notify the users also in the following cases:

- a forgetten or passed activity;
- a pending event which is not assigned yet with a scheduling of reminders;
- a shared note by another user.

This could be a great improvement that allows users to be more aware of what is happening inside the application. However, this could also result in an overloading of notifications. For this reason, it could be possible to build customizable notification settings that would allow users to adjust the frequency and type of notifications according to their preferences.

Furthermore, a notification could be made more informative by including contextual navigation or additional relevant informations with the addition of a payload. This would allow users to navigate to the appropriate section of the app or receive important details directly from the notification banner, improving the overall user experience within the application.

6.2.2 Media

Actually, the application uses only images as media type. It would be valuable to explore wheter integrating different types of media might positively impact the user experience or not. For instance, in the tips section, incorporating a video could provide a clear demonstration of how to properly execute an exercise, while adding an audio component could enhance the overall athmosphere and motivation for the exercise. Providing with these media types could enhance user engagement and effectiveness of actual features.

6.2.3 Chatbot and live chat

During the screen presentations, the personified turtle obtained significant attention and it opened to the idea of creating a chatbot with the turtle as main character.

In particular, the chatbot could guide users through various features of the application, answer questions and offer support. This could enhance user engagement by providing a more personalized experience.

On the other side, since it is an AI tool, so non-deterministic, the answer could not give relevant results to the users if they asks for something very specific or the chatbot is not trained to. For this reason, we thought it might be important to flank the chatbot with a live chat in which some operators could answer only to questions that the chatbot is not able to.

6.2.4 Emergency call

Finally, the meeting concluded with a discussion on what happens if the patient has an imminent difficulty. In order to address this, it could be useful to have a widget, which can be positioned also in the lock screen, allowing the user to ask for immediate assistance. The request could be a notification or a call, but this should be defined by testing it in a real-life scenario. Then a list of availability could be inserted, ensuring that if one caregiver is unavailable, the next person on the list is alerted. This list could be personalized on a daily basis. For example, let Bob, Alice and Charlie be the caregivers and let Dave be the patient. The list of priority could be defined like this:

Monday	Tuesday	Wednesday	Thursaday	Friday	Saturday	Sunday
Alice	Bob	Charlie	Charlie	Alice	Bob	Alice
Bob	Charlie	Alice	Alice	Bob	Charlie	Charlie
Charlie	Alice	Bob	Bob	Charlie	Alice	Bob

So:

- if Dave needs help on Monday, Alice is alerted first and, if she does not answer, Bob is alerted and so on;
- if Dave needs help on Wednesday, Charlie is alerted first and, if she does not answer, Alice is alerted and so on.

It is important to underline that this feature could be highly valuable, but it may also have a limitation. In fact, patients' behavior can be very different. For instance, some patients might never use the feature, while others might rely on it too much. Therefore, it is crucial to develop solutions that address potential issues related to the different patients' characteristics.

6.3 Final considerations

In summary, this work covered some gaps present in existing caregiving applications by creating a cross-platform mobile application with an intuitive and engaging design. In particular, our focus was on building a system that both enhances the user experience and reduces the workload of caregivers. By prioritizing user-friendly features and a simple interface, we aimed to develop a tool that improves both efficiency and ease of use, supporting caregivers in their daily responsibilities. The significance of this project extended over the technical aspects since it kept an eye on the critical role caregivers play in society.

Moreover, reducing the burden on caregivers has huge implications for the overall quality of their life. When caregivers are equipped with tools that improve their efficiency and reduce stress, they can dedicate more time and energy to other important activities, such as social interactions and self-care, which are very important for all the people.

In conclusion, this work represents an important step in providing essential support to caregivers having patients with Alzheimer, enhancing their efficiency, reducing stress and improving the quality of care provided. We addressed the limits of existing applications by using a user-centered approach and we hope to make a meaningful and lasting impact on the lives of both caregivers and their loved ones.

Appendix A

In this section we report the devices used to develop and test the application and the licenses under which the used packages are subjected. In addition, there are other useful informations for the future development.

Devices configurations

For the Android environment, we used both a virtual emulator and a physical device:

- **Virtual emulator:** *Pixel 3a XL*, screen size 6", SDK level API 33 and Android 13;
- **Physical device:** *Pixel 7*, screen size 6.3" and Android 13.

Meanwhile for the iOS environment, were not able to test because of problems arised with the university's Apple developer account. On the other side, we should expect that the Apple version works as the Android one.

The PC hardware configurations used for the development are the following:

- **Hardware Model:** Micro-Star International Co., Ltd. Modern 15 A11M;
- **RAM:** 16 GB;
- **Processor:** 11th Gen Intel® Core™ i5-1135G7 @ 2.40GHz x 8;
- **Graphics:** Mesa Intel® Xe Graphics (TGL GT2);
- **OS:** Windows 11 Pro, 64-bit.

The PC software configurations used for the development are the following

- **Flutter:** 3.22.2;
- **Dev Tools:** 2.34.3;
- **Dart SDK:** 3.4.3;
- **Android Studio:** Jellyfish 2023.3.1.

Licenses

The packages used for the development of the application have the following licences:

- **MIT:** it is a permissive software license created at the Massachusetts Institute of Technology (MIT) in the late 1980s. It is considered a permissive license, as it puts very few restrictions on reuse and therefore has high license compatibility [78];
- **Apache-2.0:** it is a permissive free software license written by the Apache Software Foundation (ASF). It allows users to use the software for any purpose, e.g., distribute it, modify it and distribute modified versions of the software under the terms of the license, without concern for royalties. The ASF and its projects release their software products under the Apache License. The license is also used by many non-ASF projects [38];
- **BSD-3-Clause:** it is a licence where redistribution and use in source and binary forms, with or without modification, are permitted under specific conditions. In particular, redistributions of source code or binary form must retain the above copyright notice, this list of conditions and the disclaimer. Additionally neither the name of the copyright holder nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission [46].

So, it is important to underline that NetCare, since it is using libraries which are under the BSD-3-Clause license, is subjected to the obligations of these packages if it will be redistributed in the future.

Firestore configuration

As we have already mentioned in Section 4.4.2, we used Firebase Firestore as database for saving the informations. We properly organised data in collections of documents, each of them having a specified number o fields. Obvioulsly, a document can have one or more subcollection, ensuring a structured, hierarchical organization. By looking at the following, we can observe how the database was organised.

Notation and comments:

- nesting is used to better understand the hierarchical level of the elements;
- each field has a String name and value which can be any type available in Firestore;
- greater and less symbols <> are used to represesent a collection;
- squared brackets [] are used to represent arrays. There are not pairs but only values which are noted as VALUE_X => Type;
- rounded brackets () are used to represent a group offields in a document. A document can have a subcollection as one of its fields;
- curly brackets {} are used to represent maps.

Caregiver Tips

Caregiver Tips collection is represented as follows:

```
caregiverTips: <
    uuidCaregiverTip0: (
        category: Int,
        description: String,
        duration: Int,
        media: [
            VALUE_0 => {position: Int, url: String}, // text offset + link
            ↪  media
            VALUE_1 => {position: Int, url: String},
            ...
        ],
        title: String
    ),
    uuidCaregiverTip1: (...),
    ...
>
```

Codes

Codes collection is represented as follows:

```
codes <
    otp: (
        valid: <
            uuidCode0: (value: String),
            uuidCode1: (value: String),
            ...
        >,
        invalid: <
            uuidCode0: (value: String),
            uuidCode1: (value: String),
            ...
        >
    )
>
```

Patient Tip's Care

Patient Tip's Care collection is represented as follows:

```
patientCareTips: <
    uuidPatientTip0: (
        category: Int,
        description: String,
        phase: Int,
        media: [
            VALUE_0 => {position: Int, url: String}, // text offset + link
            ↪  media
            VALUE_1 => {position: Int, url: String},
            ...
        ],
        title: String
    ),
    uuidPatientTip1: (...),
    ...
>
```

Rooms

Rooms collection is represented as follows:

```
rooms: <
    uuidRoom0: (
        name: String,
        users: [
            uuidUser0: String,
            uuidUser1: String,
            ...
        ],
        uuidCreator: String,
        calendar: <
            uuidEvent0: (
                assignment: String,
                description: String,
                endDate: String,
                isActivity: Bool,
                isActivityDone: Bool,
                media: [
                    VALUE_0 => String, // link media
                    VALUE_1 => String,
                    ...
                ],
                nameAssignment: String,
                startDate: String,
                surnameAssignment: String,
                title: String
            ),
            uuidEvent1: (...),
            ...
        >
    ),
    uuidRoom1: (...),
    ...
>
```

In order to share the calendar between users, we directly nested the *calendar*.

Users

Users collection is represented as follows:

```
users: <
    uuidUser0: (
        apnsToken: String,
        email: String,
        fcmToken: String,
        isCaregiver: Bool,
        isRoomCreator: Bool,
        name: String,
        pendingInvites: [
            VALUE_0 => String, // uuidRoom
            VALUE_1 => String
            ...
        ],
        sharedNotes: [
            VALUE_0 => {uuidNote: String, uuidUserShareFrom: String}, //
            ↪  uuidNote + uuidSharer
            VALUE_1 => {uuidNote: String, uuidUserShareFrom: String}
            ...
        ],
        surname: String,
        notes: <
            uuidNote1: (
                content: String,
                hashtag: String,
                indexColor: Int,
                lastModifiedDate: String,
                media: [
                    VALUE_0 => {position: Int, url: String},
                    VALUE_1 => {position: Int, url: String},
                    ...
                ],
                title: String
            ),
            uuidNote0: (...),
            ...
        >
    )
    uuidUser1: (...),
    ...
>
```

To allow users to have their own personal notes, we placed the *notes* subcollection as a field in the user document, but, in order to allow to receive the shared ones, we added the *shareNotes* out of the notes subcollection.

Bibliography

[1] T. Halbach et al. *Mobile application for supporting dementia relatives: A case study*. Vol. 256. 2018. DOI: 10.3233/978-1-61499-923-2-839 (cit. on pp. 9, 10).

[2] Gaggi Ombretta. *Mobile programming course* (cit. on pp. 83–85).

[3] Andrew J. Elliot. *Color and psychological functioning: a review of theoretical and empirical work*. Vol. 5. 2015. DOI: 10.3389/fpsyg.2015.00368 (cit. on p. 58).

[4] Ellen Leslie Brown et al. *CareHeroes Web and Android^{TM} Apps for Dementia Caregivers: A Feasibility Study*. Vol. 9. 2016. DOI: 10.3928/19404921-20160229-02. URL: https://journals.healio.com/doi/abs/10.3928/19404921-20160229-02 (cit. on pp. 9, 10).

[5] Ellen Brown et al. *Smartphone-Based Health Technologies for Dementia Care: Opportunities, Challenges, and Current Practices*. Vol. 38. 2017. DOI: 10.1177/0733464817723088 (cit. on pp. 9–11, 13).

[6] Vivian Tran Louise I. R. Castillo and Thomas Hadjistavropoulos. *Are mobile apps meeting the needs of caregivers of people living with dementia? An evaluation of existing apps for caregivers*. Vol. 28. 2024. DOI: 10.1080/13607863.2023.2177832 (cit. on pp. 9, 13).

[7] Hannah Liane Christie et al. *Lessons Learned From an Effectiveness Evaluation of Inlife, a Web-Based Social Support Intervention for Caregivers of People With Dementia: Randomized Controlled Trial*. Vol. 5. 2022. DOI: 10.2196/38656 (cit. on p. 19).

[8] Alieske E.H. Dam et al. *Process evaluation of a social support platform 'Inlife' for caregivers of people with dementia*. Vol. 15. 2019. DOI: 10.1016/j.invent.2018.09.002 (cit. on p. 5).

[9] B.H. Davis et al. *Developing a pilot e-mobile app for dementia caregiver support: Lessons learned*. Vol. 18. 2014. URL: https://ojni.org/issues/?p=3095 (cit. on pp. 9, 10).

[10] Marina Sala-González et al. *Mobile Apps for Helping Informal Caregivers: A Systematic Review*. Vol. 18. 2021. DOI: 10.3390/ijerph18041702. URL: https://www.mdpi.com/1660-4601/18/4/1702 (cit. on pp. 9, 10).

[11] Milena Guessi, Amika Shah, and Emily Seto. *Smartphone applications for informal caregivers of chronically ill patients: a scoping review*. Vol. 5. 2022. DOI: 10.1038/s41746-022-00567-z (cit. on pp. 9, 11, 12).

[12] Lesley Lubos. *The Role of Colors in Stress Reduction*. Vol. 5. 2012. DOI: 10.7828/ljher.v5i2.39 (cit. on p. 58).

[13] Bethan Naunton Morgan et al. *eHealth online interventions for informal carers of people with dementia in the community: An umbrella review*. Vol. 24. 2022. DOI: https://10.2196/36727 (cit. on p. 19).

[14] Nayab Akhtar and Sana Ghafoor. *Analysis of Architectural Patterns for Android Development*. 2021 (cit. on pp. 38, 39, 41, 42).

[15] OECD. *Health at a Glance 2021*. 2021. DOI: 10.1787/ae3016b9-en. URL: https://www.oecd-ilibrary.org/content/publication/ae3016b9-en (cit. on p. 1).

[16] Robin Nunkesser. *Choosing a Global Architecture for Mobile Applications*. 2021. DOI: 10.36227/techrxiv.14212571 (cit. on pp. 38, 39, 41).

[17] YunHee Shin et al. *Effects of App-Based Mobile Interventions for Dementia Family Caregivers: A Systematic Review and Meta-Analysis*. Vol. 51. 2022. DOI: 10.1159/000524780 (cit. on pp. 9, 13).

[18] Nicholas Leigh-Hunt et al. *An overview of systematic reviews on the public health consequences of social isolation and loneliness*. Vol. 152. 2017. URL: https://api.semanticscholar.org/CorpusID:19063958 (cit. on p. 3).

[19] N.C. Tan et al. *Patient-caregiver twinned mobile phone application to promote medication adherence*. Vol. 33. 2024. DOI: 10.1177/20101058241227335 (cit. on pp. 10, 13, 27, 28).

[20] Taylor A. James, Dara James, and Linda K. Larkey. *Heart-focused breathing and perceptions of burden in Alzheimer's caregivers: An online randomized controlled pilot study*. Vol. 42. 2021. DOI: 10.1016/j.gerinurse.2021.02.006 (cit. on pp. 9, 13).

[21] Isabelle Velloze et al. *Interventions to Reduce Loneliness in Caregivers: An Integrative Review of the Literature*. Vol. 311. 2022. DOI: 10.1016/j.psychres.2022.114508 (cit. on p. 3).

[22] Angel H. Wang et al. *Beyond instrumental support: Mobile application use by family caregivers of persons living with dementia*. Vol. 21. 2022. DOI: 10.1177/14713012211073440 (cit. on pp. 10, 13).

[23] L. Wilms and D. Oberfeld. *Color and emotion: effects of hue, saturation and brightness*. Vol. 82. 2018. DOI: 10.1007/s00426-017-0880-8 (cit. on p. 58).

[24] Lori Wozney et al. *Commercially Available Mobile Apps for Caregivers of People With Alzheimer Disease or Other Related Dementias: Systematic Search*. Vol. 1. 2018. DOI: 10.2196/12274 (cit. on pp. 9, 11).

[25] Javier Yanguas, Sacramento Pinazo-Henandis, and Francisco Tarazona-Santabalbina. *The complexity of loneliness*. Vol. 89. 2018. DOI: 10.23750/abm.v89i2.7404 (cit. on p. 3).

[26] *Clean Code: A Handbook of Agile Software Craftsmanship*. 2008, p. 464 (cit. on p. 44).

[27] *General Data Protection Regulation EU*. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679 (cit. on pp. 28, 91).

[28] *A Walk through Dementia*. URL: https://www.alzheimersresearchuk.org/ (cit. on pp. 5, 13).

[29] *AI Act document*. URL: https://www.europarl.europa.eu/doceo/document/TA-9-2024-0138_EN.pdf (cit. on p. 58).

[30] *AlayaCare App Store*. URL: https://apps.apple.com/it/app/alayacare/id1030754584 (cit. on p. 15).

[31] *AlayaCare Play Store*. URL: https://play.google.com/store/apps/details?id=com.alayacare.careworkerapp&hl=en (cit. on p. 15).

[32] *Alzheimer's Disease Caregivers, Alzheimer Association, March, 2024*. URL: https://portal.alzimpact.org/media/serve/id/5a31f42654048 (cit. on p. 2).

[33] *Alzheimer Association daily care plan*. URL: https://www.alz.org/help-support/caregiving/daily-care/daily-care-plan (cit. on p. 28).

[34] *Android architecture documentation*. URL: https://developer.android.com/topic/architecture (cit. on p. 43).

[35] *Android Studio IDE*. URL: https://developer.android.com/studio/intro (cit. on p. 30).

[36] *Annual global life expectancy at select ages from 1950 to 2022, with projections until 2100 [Graph], UN DESA, July 30, 2022*. URL: https://www.statista.com/statistics/1460165/global-life-expectancy-by-age-historical/ (cit. on p. 1).

[37] *AOT vs JIT compilation*. URL: https://medium.com/@sivakishore.teru/aot-vs-jit-compilation-c99e29cc733d (cit. on p. 31).

[38] *Apache License*. URL: https://en.wikipedia.org/wiki/Apache_License (cit. on p. 95).

[39] *Choosing Android Architectures: MVC, MVP, MVVM, Clean Architecture*. URL: https://medium.com/@KodeFlap/choosing-android-architectures-mvc-mvp-mvvm-clean-architecture-and-mvi-8ad2a43f7f9b (cit. on p. 38).

[40] *Unit Testing and the Arrange, Act and Assert (AAA) Pattern*. URL: https://medium.com/@pjbgf/title-testing-code-ocd-and-the-aaa-pattern-df453975ab80 (cit. on p. 88).

[41] *Average life expectancy in selected countries as of 2022 [Graph], OECD, December 12, 2023*. URL: https://www.statista.com/statistics/236583/global-life-expectancy-by-country/ (cit. on p. 1).

[42] *Bloc concepts*. URL: https://bloclibrary.dev/bloc-concepts/ (cit. on p. 47).

[43] *Flutter State Management: BLoC Pattern*. URL: https://medium.com/@aaron.chu/flutter-state-management-bloc-pattern-9cd6011c699 (cit. on p. 43).

[44] *Flutter bloc library*. URL: https://pub.dev/packages/bloc (cit. on p. 47).

[45] *Bloc Test package*. URL: https://pub.dev/packages/bloc_test (cit. on p. 87).

[46] *BSD-3-Clause License*. URL: https://opensource.org/license/bsd-3-clause (cit. on p. 95).

[47] *MVC, MVP, MVVM and VIPER architecture patterns*. URL: https://blog.bytebytego.com/p/ep49-api-architectural-styles (cit. on pp. 38, 40–42).

[48] *Caregiver Mobile App Store*. URL: https://apps.apple.com/it/app/caregiver-mobile/id1510594355 (cit. on p. 18).

[49] *Carely Caregiving.com*. URL: https://www.caregiving.com/ (cit. on p. 5).

[50] *The Clean Code Blog - The Clean Architecture*. URL: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html (cit. on pp. 44, 45).

[51] *Cloud Functions version comparison*. URL: https://firebase.google.com/docs/functions/version-comparison (cit. on p. 80).

[52] *WebAIM Contrast Checker*. URL: https://webaim.org/resources/contrastchecker/ (cit. on p. 83).

[53] *Dart website*. URL: https://dart.dev/ (cit. on p. 31).

[54] *Delta operations*. URL: https://quilljs.com/docs/delta/ (cit. on p. 56).

[55] *Dementia Talk App*. URL: https://www.cabhi.com/completed-project-summaries/dementia-talk-app/ (cit. on pp. 5, 13).

[56] *Firebase website*. URL: https://firebase.google.com/ (cit. on p. 33).

[57] *Firebase Authentication documentation*. URL: https://firebase.google.com/docs/auth (cit. on p. 33).

[58] *Cloud Functions for Firebase documentation*. URL: https://firebase.google.com/docs/functions (cit. on p. 36).

[59] *Firebase Firestore documentation*. URL: https://firebase.google.com/docs/firestore (cit. on p. 34).

[60] *Firebase Cloud Messaging documentation*. URL: https://firebase.google.com/docs/cloud-messaging (cit. on p. 36).

[61] *Firebase Realtime Database documentation*. URL: https://firebase.google.com/docs/database (cit. on p. 34).

[62] *Choose a Database: Cloud Firestore vs Realtime Database*. URL: https://firebase.google.com/docs/database/rtdb-vs-firestore (cit. on p. 35).

[63] *Cloud Storage for Firebase documentation*. URL: https://firebase.google.com/docs/storage (cit. on p. 35).

[64] *Flutter website*. URL: https://flutter.dev/ (cit. on p. 32).

[65] *Flutter accessibility*. URL: https://docs.flutter.dev/ui/accessibility-and-internationalization/accessibility (cit. on p. 85).

[66] *Bloc concepts*. URL: https://bloclibrary.dev/flutter-bloc-concepts/ (cit. on p. 52).

[67] *Flutter Quill package*. URL: https://pub.dev/packages/flutter_quill (cit. on pp. 55, 72).

[68] *An introduction to unit testing*. URL: https://docs.flutter.dev/cookbook/testing/unit/introduction (cit. on p. 88).

[69] *Freezed package*. URL: https://pub.dev/packages/freezed (cit. on p. 54).

[70] *GetIt library*. URL: https://pub.dev/packages/get_it (cit. on p. 46).

[71] *Global Health Caregiving Market Size, Share, Trends, COVID-19 Impact & Growth Analysis Report - Segmented By Care Type, End-User and Region (North America, Europe, APAC, Latin America, Middle East and Africa) – Industry Forecast (2024 to 2029)*. URL: https://www.marketdataforecast.com/market-reports/health-caregiving-market (cit. on p. 1).

[72] *Hibi App Store*. URL: https://apps.apple.com/it/app/hibi-care-companion/id6448462377 (cit. on p. 17).

[73] *Hibi Play Store*. URL: https://play.google.com/store/apps/details?id=com.hibi.app&hl=en (cit. on p. 17).

[74] *IanaCare App Store*. URL: https://apps.apple.com/it/app/ianacare-caregiving-support/id1441737626 (cit. on p. 16).

[75] *IanaCare Play Store*. URL: https://play.google.com/store/apps/details?id=com.ianacare.ianacare&hl=en (cit. on p. 16).

[76] *Json Serializable package*. URL: https://pub.dev/packages/json_serializable (cit. on p. 54).

[77] *Material Design documentation*. URL: https://material.io/guidelines/material-design/introduction.html (cit. on pp. 58, 81).

[78] *MIT License*. URL: https://en.wikipedia.org/wiki/MIT_License (cit. on p. 95).

[79] *Mockito package*. URL: https://pub.dev/packages/mockito (cit. on p. 87).

[80] *Service Locator Pattern*. URL: https://salmanbediya.medium.com/getit-simplifying-dependency-injection-with-service-locator-pattern-in-dart-and-flutter-62a2d7d105b8 (cit. on p. 46).

[81] *Sizer package*. URL: https://pub.dev/packages/sizer (cit. on p. 58).

[82] *SOLID principles*. URL: https://staff.cs.utu.fi/~jounsmed/doos_06/material/DesignPrinciplesAndPatterns.pdf (cit. on p. 43).

[83] *Architecting iOS Apps with VIPER*. URL: https://www.objc.io/issues/13-architecture/viper/ (cit. on pp. 42, 43).

[84] *WCAG Mobile accessibility mapping*. URL: https://www.w3.org/TR/mobile-accessibility-mapping/ (cit. on p. 81).